Conceptual Framework and Methodology for Analysing Previous Molecular Docking Results

Temelkovski, D.

University of Westminster
Faculty of Science and Technology
Department of Computer Science

Conceptual Framework and Methodology for Analysing Previous Molecular Docking Results

Damjan Temelkovski

Submitted in partial fulfilment of the requirements of the University of Westminster for the degree of Doctor of Philosophy

March 2019

Abstract

Modern drug discovery relies on *in-silico* computational simulations such as molecular docking. Molecular docking models biochemical interactions to predict where and how two molecules would bind. The results of large-scale molecular docking simulations can provide valuable insight into the relationship between two molecules. This is useful to a biomedical scientist before conducting *in-vitro* or *in-vivo* wet-lab experiments. Although this field has seen great advancements, feedback from biomedical scientists shows that there is a need for storage and further analysis of molecular docking results. To meet this need, biomedical scientists need to have access to computing, data, and network resources, and require specific knowledge or skills they might lack.

Therefore, a conceptual framework specifically tailored to enable biomedical scientists to reuse molecular docking results, and a methodology which uses regular input from scientists, has been proposed. The framework is composed of 5 types of elements and 13 interfaces. The methodology is light and relies on frequent communication between biomedical sciences and computer science experts, specified by particular roles. It shows how developers can benefit from using the framework which allows them to determine whether a scenario fits the framework, whether an already implemented element can be reused, or whether a newly proposed tool can be used as an element.

Three scenarios that show the versatility of this new framework and the methodology based on it, have been identified and implemented. A methodical planning and design approach was used and it was shown that the implementations are at least as usable as existing solutions. To eliminate the need for access to expensive computing infrastructure, state-of-the-art cloud computing techniques are used.

The implementations enable faster identification of new molecules for use in docking, direct querying of existing databases, and simpler learning of good molecular docking practice without the need to manually run multiple tools. Thus, the framework and methodology enable more user-friendly implementations, and less error-prone use of computational methods in drug discovery. Their use could lead to more effective discovery of new drugs.

Acknowledgements

I would like to express my gratitude to Prof Tamas Kiss, my Director of Studies, for his remarkable support and guidance. I would also like to thank my Second Supervisors, Dr Pamela Greenwell and Prof Gabor Terstyanszky, for sharing their knowledge and advice in biomedical sciences and computer science. I have been very lucky to have such a devoted supervisory team that was always ready to discuss my research enthusiastically. It was a privilege to learn from, and be guided by them.

I would like to thank the University of Westminster for providing me with a full scholarship, equipment and conditions without which my research project would not be possible. I would like to thank all the technical, facilities, and library staff. In particular, I would like to thank all the colleagues and friends from the computational research offices on the 7^{th} floor and the biological office on the 4^{th} floor. Thank you Hans, Gregoire, Noam, Hannu, Juha, thank you Zoltan and all the kind people at SZTAKI.

I would not have come this far in my PhD journey without the love and support of my family. Thank you Svetlana, Venko, Joana, and Luisa for always being by my side even if at times we were far from each other. I would like to thank all my friends in Macedonia, in London, and around the world. Thank you Goodenough College for being my home away from home where many great friendships started.

Decleration of Originality

I declare that the present work was carried out in accordance with the Guidelines and Regulations of the University of Westminster. The work is original except where indicated by special reference in the text.

The submission as a whole or part is not substantially the same as any that I previously made or am currently making, whether in published or unpublished form, for a degree, diploma or similar qualification at any university or similar institution.

Until the outcome of the current application to the University of Westminster is known, the work will not be submitted for any such qualification at another university or similar institution. Any views expressed in this work are those of the author and in no way represent those of the University of Westminster.

Signed:

Date:

Glossary

Aromatic molecule An aromatic molecule contains a cyclic ring of atoms that provide high stability, most commonly a benzene ring which is a hexagonal hydrocarbon made of 6 carbon atoms (e.g. C_6H_6).

Da Dalton (Da), also known as the unified atomic mass unit (amu), is a standard unit of mass of small entities such as atoms and molecules. One Da equals one twelfth of the mass of Carbon-12 (^{12}C) or 1.660539×10^{-27} kg.

Evolutionary distance The evolutionary distance between two proteins is usually calculated as the number of amino acid substitutions between two homologous (evolutionarily related) proteins.

Hydrophobic/Hydrophilic Molecules or sets of atoms that repel the water molecule are called hydrophobic. Molecules that bond with the water molecule are called hydrophilic. A large protein can have a section that is hydrophobic and a section that is hydrophilic.

NMR spectroscopy Nuclear Magnetic Resonance (NMR) spectroscopy is used to calculate the structure of a molecule and its conformation in solution. A conformation is the spatial arrangement of atoms in a molecule.

Polar molecule A polar molecule has partial positive charges on one side and partial negative charges on another side due to polar bonds - (H_2O) is a typical example.

Tanimoto Coefficient A measure of similarity between two binary variables, the Tanimoto coefficient is equal to $\frac{\sum_{i=1}^{k}(a_i \times b_i)}{\sum_{i=1}^{k}(a_i^2)+\sum_{i=1}^{k}(b_i^2)-\sum_{i=1}^{k}(a_i \times b_i)}$. It is commonly used to compare two molecules that have been represented by their chemical fingerprint, which is a vector with values 0 or 1 that describes the molecular structure of small molecules. When comparing two sets, this type of similarity measure is known as the Jaccard Index and is equal to: $\frac{A \cap B}{A \cup B}$.

X-ray crystallography X-ray crystallography uses the diffraction of X-rays from a crystallised molecule in order to determine the three-dimensional structure of the molecule. The diffraction pattern obtained from the X-rays scattering off the crystal is used to calculate the density of electrons and deduce the structure.

Contents

List of Tables

List of Figures

List of Abbreviations

ADMETox	Absorption, Distribution, Metabolism, and Excretion, Toxicity, page 68
ADS	Additional Data Source, page 56
ADT	AutoDock Tools, page 19
API	Application Programming Interface, page 27
ASM	Application Specific Module, page 27
AT	Additional Tool, page 55
CADD	Computer-Aided Drug Design, page 2
CDK	Chemistry Development Kit, page 31
CML	Chemical Markup Language, page 29
config	Configuration, page 19
DM	Decision Maker, page 56
DUD	Directory of Useful Decoys, page 15
EC2	Elastic Compute Cloud (Amazon), page 23
GOLD	Genetic Optimisation for Ligand Docking, page 17
GUI	Graphical User Interface, page 19
gUSE	grid User Support Environment, page 25
HPC	High Performance Computing, page 21

HTS	High Throughput Screening, page 2
IaaS	Infrastructure-as-a-Service, page 22
KACR	Kepler Analytical Component Repository, page 30
LBVS	Ligand-Based Virtual Screening, page 2
LGA	Lamarkian Genetic Algorithm, page 18
M & S	Modelling & Simulation, page 28
MD	Molecular Dynamics, page 17
MDE	Molecular Docking Environment, page 55
MDRR	Molecular Docking Results Repository, page 55
MM/GBSA	Molecular Mechanics/Generalised Born Surface Area, page 66
MoSGrid	Molecular Simulation Grid, page 29
MPI	Message Passing Interface, page 23
MSML	Molecular Simulation Markup Language, page 29
PaaS	Platform-as-a-Service, page 22
PBS	Portable Batch System, page 20, 21
PDB	Protein Data Bank, page 10
PUG	Power User Gateway (PubChem), page 119
QC	Quantum Chemistry, page 29
QSAR	Quantitative Structure-Activity Relationship, page 14
RMSD	Root-Mean-Square Deviation, page 12
SaaS	Software-as-a-Service, page 22
SBVS	Structure-Based Virtual Screening, page 2
SGE	Sun Grid Engine, page 20, 21

SME	Small and Medium sized Enterprise / Simulation in Manufacturing and Engineering, page 28
SMILES	Simplified Molecular-Input Line-Entry System, page 10
SPP	Semantic Provenance Processor, page 30
sTC	scaled Tanimoto Coefficient, page 15
SZDG	SZTAKI Desktop Grid, page 42
TC	Tanimoto Coefficient, page 15
TV	Trichomonas vaginalis, page 40
UML	Unified Modelling Language, page 79
UoW	University of Westminster, page 1
VS	Virtual Screening, page 2
WS-PGRADE	originally Web Services - Parallel Grid Runtime and Developer Environment, page 26

Chapter 1

Introduction

1.1 Motivation and Objectives

Throughout history, serendipity has played a major role in the discovery of medicines and medical drugs. Perhaps the most notable case in the 20^{th} century is the revolutionary discovery of penicillin [1] in 1927 by Sir Alexander Fleming. An alumnus of the Regent Street Polytechnic, which eventually became the University of Westminster (UoW), Fleming discovered the anti-bacterial properties of a type of mould. The mould had grown serendipitously, as a contaminant on a Petri dish seeded with bacteria while he was away on holiday [2].

Many drugs have been discovered using "classical pharmacology". Here, the effects (functional activity) of a substance on an organism or cell are determined, either by serendipity or through screening, before the biological target for this interaction is identified. One example is the discovery of Tamsulosin, a drug used to treat benign prostatic hyperplasia. Scientists at Yamanouchi Pharmaceutical firstly discovered the effect of Tamsulosin on the prostate, before conducting additional experiments to conclude that this is due to its high affinity for the *alpha-1B adrenergic receptor* [3].

In an alternative approach, known as "reverse pharmacology" or "rational drug design", scientists do the opposite. They start by identifying a biological target, hypothesising that its modification will result in a therapeutic effect. This hypothesis can be tested by screening a large library of potential drug candidates and assessing the interaction with the target. Finally, the successful candidate of the screening is tested in living organisms to show the functional activity [3].

Nowadays, the process of screening drug candidates would consist of High Throughput Screening (HTS), a method using automated laboratory equipment which can quickly

assay the interaction between drug candidates and biological targets. The development of modern robotics has enabled HTS facilities to screen hundreds and even thousands of drug candidates per day. However, the cost of the assays and the requirement to source many, potentially expensive, drugs makes this technology available only to well-funded laboratories.

On the other hand, developments in bioinformatics have given rise to the concept of Virtual Screening (VS). VS aims at producing results analogous to HTS, but instead of using automatic assays, it relies on using bioinformatics to calculate or estimate the interaction between the drug candidate and the biological target. In comparison to HTS, VS is an inexpensive method that enables scientists to screen millions of molecules and identify a small amount of candidates to test in the "wet lab". In general, in this kind of interaction, the drug candidate is known as a "ligand", while the biological target is a "receptor". VS and other similar bioinformatics techniques form part of a field known as Computer-Aided Drug Design (CADD), and are often used in rational drug design.

There are two different types of VS: Ligand-Based Virtual Screening (LBVS), and Structure-Based Virtual Screening (SBVS). In order to conduct LBVS, scientists may have a description of a known ligand that binds to the receptor, and then search for a ligand similar to it. Alternatively, scientists use the descriptions of many known ligands to create a "pharmacophore model", a hypothetical substance which contains elements from the known ligands. Then, they would search through a large library of existing molecules for one that is similar to the pharmacophore model.

Conversely, SBVS uses a predetermined description of the receptor's 3D structure and a large library of ligands in order to calculate the most likely ligand that binds to the receptor. In order to produce this calculation, SBVS uses a technique known as "molecular docking". Molecular docking is the term used for a software simulation that predicts the interaction between two molecules, ligand and receptor, by calculating how likely it is for them to bind, based on their 3D structures. Molecular docking and other structure-based CADD techniques have been used to discover the drug Aliskiren (Rasilez) [4]. Other drugs that have utilised CADD in the drug discovery process include: Dorzolamide (Trusopt), Zanamivir (Relenza), Nelfinavir (Viracept), and at least 6 others [5].

Even though molecular docking results are just an estimate that needs to be confirmed by additional analysis or laboratory experiments, the knowledge they provide can be key to the discovery of new drugs. For instance, molecular docking is part of drug discovery efforts to treat the rare genetic disease *N-Glycanase deficiency*, caused by a mutation of the gene NGLY1. SBVS has been used to dock 13 FDA-approved drugs to a proposed biological target, the *ENGase inhibitor* [6]. VS and HTS have provided nine ligands that bind to the N-Glycanase protein, and may lead to potential therapeutic applications [7]. Matthew

Might, one of the many researchers in this area, and parent of the first documented patient with N-Glycanase deficiency [8], is a professor in Computer Science (and creator of a very useful guide for PhD students [9]). This further emphasises the interconnectivity between computational techniques, such as molecular docking and VS, and drug discovery. Furthermore, there are many other disciplines where these techniques are important. For instance, molecular docking was recently used to assess the impact of veterinary medicines on non-target organisms and the environment [10].

Inspired by such research efforts, this PhD thesis aims at improving the current landscape by enabling biomedical scientists to use molecular docking and virtual screening simulations for more interesting projects, while making the development of computer systems based on these simulations easier for software developers. In the remainder of the thesis the term VS will be used to signify SBVS, particularly large-scale molecular docking simulations.

Best practices can be improved by providing an environment to make it easier for biomedical scientists to use molecular docking and VS scientific simulations, in order to broaden their applicable use in further domains, and to extend their execution environment onto a far wider scale of computing infrastructures, including cloud computing. Furthermore, providing a docking result repository where scientists could share their results, would enable additional conclusions based on prior docking results, thereby improving the current landscape of molecular docking.

Chapter 2 provides the required background for the research shown in this thesis. VS simulations are computationally demanding and require complex computer infrastructure to produce results in a reasonable time. High Performance Computing (HPC) clusters have been traditionally used, but recently there is a growing use of cloud computing for scientific simulations. A research gap exists in this area. Chapter 3 focuses on filling this gap by proposing a concept for extending popular desktop applications with cloud computing capabilities. If biomedical scientists had seamless access to run large-scale simulations directly from their favourite desktop application, using cloud computing, they would no longer need access to HPC clusters.

The remainder of the thesis focuses on a second gap that has been identified, namely, that docking results are currently not shared and that enabling a shared repository would be a useful tool to better foster collaboration and reuse. This is important for many disciplines, including drug discovery. A repository would enable scientists to make additional conclusions based on the simulation results they have obtained in the past, or results obtained by other scientists. For instance, it would prevent repeating the same molecular docking simulation and it would facilitate learning. Extending existing tools to this effect in an *ad-hoc* manner would be too difficult. In order to aid software developers in creating computer

systems based on storing or using previously stored molecular docking results, this thesis suggests a more formalised framework and a specific software development methodology.

The need for storing and sharing molecular docking results has been already identified. A survey of a bioinformatics community has shown that nearly ¾ of the community would share their input and results files in a repository after they have published their research, while almost 90% would share the tools and workflows they have used [11]. To further examine the need for a molecular docking result repository, a set of interviews with biomedical scientists have been conducted as part of this thesis. Chapter 4 presents a generic conceptual framework for software systems that analyse molecular docking results, which has been defined based on the interviews and a literature review of existing systems (Chapter 5). A specific software development methodology which includes the use of the framework is proposed and explained in Chapter 7. The benefits of the framework and the methodology are explored in Chapters 8 and 9 by producing prototype implementations of three scenarios that analyse previous molecular docking results.

1.2 Contributions

This PhD thesis explores ways to improve how biomedical scientists use molecular docking and virtual screening simulations. It proposes a way to make the development of computer systems based on these simulations easier for software developers.

While supporting biomedical scientists in conducting bioinformatics simulations, the candidate realised that there is a gap in the tools currently used for VS. A popular desktop application can help users run VS simulations on an HPC cluster, but biomedical scientists that do not have access to clusters need a VS tool that uses cloud computing. The lack of a vendor-independent way to extend desktop applications was an inspiration for the development of the first contribution of this thesis: a generic concept for extending desktop applications with cloud computing capabilities. Using this concept, the VS desktop application can be extended and the simulations can be conducted on clouds.

Although this improved the accessibility of VS simulations drastically, it was evident that scientists did not store and share docking results. If publicly available, these docking results can be used by other scientists through software systems that make conclusions or decisions based on previous docking results. The need for such systems was explored through interviews and a literature review was used to fortify the second contribution of this thesis: a generic conceptual framework for systems that use docking results. The third contribution, a software development methodology that proposes a way to use the framework, was required to help software engineers in the creation of this type of software

systems. In summary, the three contributions of this thesis are:

1. *Generic concept for extending desktop applications with cloud computing capabilities.*

Domain-specific desktop applications are still widely used. The generic concept proposed in this thesis aims to enable existing and well-established desktop applications to access heterogeneous cloud computing resources. One reason why desktop applications are popular is the flexible user-friendly graphical interface that they provide. This concept provides a way for desktop applications to access cloud computing resources seamlessly without major reengineering. The end-users can use an extended version of the same desktop application and the same familiar interface while leveraging the benefits of cloud computing.

The novelty of this generic concept is the suggested use of platform- and tool-independent set of services (named Cloud Access Services - CAS). The CAS should be called directly from the back-end code of the desktop application in order to integrate the cloud computing capabilities. Additionally, the CAS should provide access to a range of cloud computing resources suitable for complex application scenarios. The CAS prevent the problem of vendor lock-in, since changing one implementation of CAS for another will not incur substantial costs. The concept proposed in this thesis is generic and applicable in all domains. Using this concept, software developers can extend domain-specific desktop applications without major effort, thus providing the benefits of cloud computing (such as reducing operational costs, scalability, and elasticity) to the users. This is the biggest impact of this contribution. Chapter 3 showcases how this concept can be used for the particular domain of molecular docking and virtual screening simulations.

2. *Generic conceptual framework for software systems that use molecular docking results.*

Molecular docking simulations can predict if two molecules will bind to each other. The molecular docking results can be useful to the scientist that created them, or to other scientists. The framework proposed in this thesis facilitates the storage and sharing of molecular docking results, by simplifying the development of software systems that use previous molecular docking results.

This is a novel tool-independent conceptual framework which allows easy plugging in of specific tools. A custom-made or an existing tool can be used in a scenario as an element of the framework. The framework defines five element types and the interfaces between them. Reusability is a major part of the framework - if a tool has been used as an element in one scenario it can easily be used in another scenario. Furthermore, through the powerful formal description of elements and interfaces, the framework allows a developer to check whether an existing tool can be used. The prospective element should be described

formally and then compared to formal abstract descriptions of an appropriate element type. Similarly, developers can check whether the framework is suitable for implementing a new scenario by describing the elements and interfaces formally, and comparing it to the generic abstract description of the framework. Chapter 4 provides details about the framework including two methods used in its construction: interviews with domain scientists and review of the literature.

3. *Methodology for developing software systems that use molecular docking results, based on the framework.*

The methodology for developing complex environments that reuse and analyse previous molecular docking results complements the framework. It is a collaborative methodology that provides a guide to a team that is about to implement a scenario using the framework. The methodology clearly states the roles that members of the team can undertake and the specific sub-projects for which they need to collaborate. It emphasises the need to design and plan the development by describing the scenario according to the defined element types and interfaces of the framework. The basis of the methodology is a Role-Deliverable-Milestone diagram. A novel addition to this diagram clearly specifies that the development process is agile. The methodology provides three techniques that specify how the abstract descriptions of the framework can be used. Chapter 7 provides more details about the methodology.

Three scenarios, identified through interviews with domain scientists, have been implemented using the framework and following the methodology. They show the different capabilities the framework offers such as implementing a new scenario easily by reusing an existing element, and the ability to use several elements of the same element type or introduce a new element type. The implementations were tested to show that following such a methodical approach produces usable systems that are not cumbersome.

1.3 Publications

As a result of the work shown in this thesis, the following publications have been created:

1. Conference paper - "Molecular docking with Raccoon2 on clouds: Extending desktop applications with cloud computing", 9^{th} International Workshop on Science Gateways (IWSG 2017), 19-21 June 2017, Poznań, Poland [12].

2. Conference paper - "A generic framework and methodology for implementing science gateways for analysing molecular docking results", 10^{th} International Workshop on Science Gateways (IWSG 2018), 13-15 June 2018, Edinburgh, UK [13].

3. Journal article - "Extending molecular docking desktop applications with cloud computing support and analysis of results", Future Generation Computer Systems, vol. 97, Special issue on Science Gateways 2017, pp. 814-824, 2019.

4. Journal article - "Building science gateways for analysing molecular docking results using a generic framework and methodology", currently under review.

Publication 1 describes the generic concept for extending domain-specific desktop applications with cloud computing capabilities, and the extension of the VS tool Raccoon2 (shown in Chapter 3). Publication 2 introduces the conceptual framework, methodology and the implementation of Scenario 1, which are described in more detail in Chapters 4, 7, and 8. Publication 3 is a journal article and a continuation of Publication 1, which shows how the extension of Raccoon2 can be included in complex implementations of scenarios that use previous molecular docking results. Finally, Publication 4 focuses on the usability tests which show the usability of the implementations, as detailed in Chapter 9.

Additionally, the candidate presented parts of the work as an oral presentation and poster ("Extending a virtual screening tool to run simulations on clouds", ISCB RSG UK 2^{nd} Bioinformatics Student Symposium, 7 October 2015, TGAC, Norwich), and short oral presentation and abstract ("Extending a molecular docking tool to run simulations on clouds", 7^{th} International Workshop on Science Gateways (IWSG 2015), 3-5 June 2015, SZTAKI, Budapest, Hungary).

Chapter 2

Background

2.1 Introduction

Several existing research areas or currently used conventions are important for any system that uses bioinformatics tools. This thesis explores tools that use the three-dimensional structure of molecules as input. Often the results of a bioinformatics analysis are only as good as the input files. Therefore, it is helpful to provide more details about the types of molecules and the way that their structure is described for computers to understand. Furthermore, two important questions can be answered based on the structure of two molecules: "How structurally similar are two molecules?", and "Where and how would two molecules bind?". Two types of algorithms, structural alignment and molecular docking respectively, can answer these questions. The focus of this thesis is on molecular docking simulations. Running a large number of molecular docking simulations is a complex and computationally demanding task. It requires the use of solutions such as scientific workflows and distributed computing infrastructures (including clouds). Existing frameworks, or indeed the framework presented in this thesis, can aid in creating a system that uses molecular docking results. This thesis builds upon existing background knowledge in all the mentioned areas, which will be overviewed in this chapter. The reader may continue reading this chapter, or refer back to relevant sections while reading the remainder of the thesis.

2.2 Description of Molecules

A substance can be divided and still retains its biochemical properties. This can be done up to a certain point. The smallest group of atoms that retain these properties is called

a molecule. This thesis focuses on two types of molecules: large proteins, and small molecules that can bind to a protein and have some effect. By convention, the former will be referred to as *receptors*, and the latter as *ligands*.

Large proteins - receptors Proteins are some of the most important molecules in living organisms. Proteins play diverse roles in organisms. For instance, some are structural, others have enzymatic properties, immunological functions, or act in cell signalling. Proteins are polypeptides, a peptide being a molecule that contains several amino acids connected by peptide bonds. Amino acids are organic compounds that contain an amine group ($-NH_2$) on one end, and a carboxyl group ($-COOH$) on the other end, with a set of carbon atoms in the middle which are connected to a side chain (referred to as R). There is a specific group of 22 amino acids which feature in proteins in all life forms on Earth (21 in humans).

Being polypeptides, proteins can be described by the list of all amino acids that comprise them. This is known as the protein amino-acid chain or *sequence*. If untangled, the protein would fold back into a specific three-dimensional form. The three-dimensional form of a protein shows several distinct elements, the most common being the Alpha (α) helix and Beta (β) pleated sheet. The sequence of amino acids represents the protein's primary structure. The protein's secondary structures are the α-helices or β-sheets. The secondary structure is held together by Hydrogen bonds, giving stability. The final 3D structure of a protein is called the tertiary structure. The tertiary structure is held together by four different bonds and interactions:

- Disulphide bonds: sulphur atoms on Cysteine amino acids form a double bond (S=S).
- Ionic bonds: two oppositely charged amino acids (+ve and -ve) that are close to each other may form ionic bonds.
- Hydrogen bonds: H atoms on different amino acids form bonds between them.
- Hydrophobic and hydrophilic interactions: some amino acids are hydrophobic while others are hydrophilic. In a water based environment, a protein orientates itself with hydrophobic parts towards its centre and hydrophilic parts towards its edges.

Two or more protein subunits may come together to form a complex, this is the protein's quaternary structure (e.g. Haemoglobin has 4 subunits). Amino acids vary in size (e.g. Glycine is 75 Da whereas Tryptophan is 204 Da), may have charge (positive or negative), may be polar, hydrophobic or hydrophilic, or have ring structures (i.e. may be aromatic). A list of the amino acids that are part of human proteins is provided in Appendix F.

In practice, the protein 3D structure can be described by the coordinates of the atoms

that compose it. The Protein Data Bank (PDB [14]) is a repository of "solved" protein structures ('protein structure' often refers to the 3D structure). Solved structures are structures that have been determined using methods such as X-ray crystallography, NMR spectroscopy, or estimated using homology modelling. Homology modelling refers to modelling the structure of an unknown protein with respect to the known structure of a homologous protein. The description of coordinates can be stored in a file in the .pdb format. An example of this file would describe the protein sequence as (amino acids shown using the three-letter abbreviations):

```
SEQRES   1 A  309  MET GLN ASN ALA GLY SER LEU VAL VAL LEU GLY SER ILE
SEQRES   2 A  309  ASN ALA ASP HIS ILE LEU ASN LEU GLN SER PHE PRO THR
SEQRES   3 A  309  PRO GLY GLU THR VAL THR GLY ASN HIS TYR GLN VAL ALA
SEQRES   4 A  309  PHE GLY GLY LYS GLY ALA ASN GLN ALA VAL ALA ALA GLY
SEQRES   5 A  309  ARG SER GLY ALA ASN ILE ALA PHE ILE ALA CYS THR GLY
SEQRES   6 A  309  ASP ASP SER ILE GLY GLU SER VAL ARG GLN GLN LEU ALA
```

The exact coordinates of the atoms would be described further down in the same file (the X, Y, and Z, coordinates show in the 7^{th}, 8^{th}, and 9^{th} columns):

```
ATOM      1  N   ALA A   4      15.854  16.067  56.619  1.00 38.52           N
ATOM      2  CA  ALA A   4      15.925  14.565  56.631  1.00 38.20           C
ATOM      3  C   ALA A   4      14.555  13.990  56.933  1.00 36.20           C
ATOM      4  O   ALA A   4      13.600  14.731  57.141  1.00 36.93           O
ATOM      5  CB  ALA A   4      16.926  14.067  57.668  1.00 38.48           C
ATOM      6  N   GLY A   5      14.462  12.667  56.957  1.00 33.75           N
```

Small molecules - ligands A ligand is a small molecule which binds to another molecule. Its chemical formula, which usually has a small number of atoms, can be used to describe it. A two-dimensional drawing is commonly drawn alongside it, to represent the types of bonds between all the atoms. Another popular notation to describe molecular structure of ligands is the Simplified Molecular-Input Line-Entry System (SMILES [15]). SMILES uses ASCII symbols to represent the structure of the molecule. Theoretically, a SMILES code is the string produced when traversing the chemical formula graph as a depth-first tree (once the graph has been converted into a spanning tree). This means that there are different ways to create a SMILES code, for instance, based on where a cycle (benzene ring) will be broken up. Several algorithms enable creating the same SMILES code from the same molecular structure (known as the "canonical SMILES" code).

Analogously to the protein structure, the ligand structure can be described by the coordinates of the atoms that compose it. Popular file formats used to represent a ligand are .mol2 or .pdb. An example of a .mol2 file would describe the ligand structure as:

```
@<TRIPOS>MOLECULE
STI
   48     51      1      1      3
PROTEIN
GASTEIGER
```

```
@<TRIPOS>DICT
PROTEIN PROTEIN
@<TRIPOS>ATOM
      1 C1          14.8490    2.8316   15.5182 C.ar      1 STI1       -0.0430
      2 C6          13.7123    3.5860   15.2146 C.ar      1 STI1       -0.0497
      3 C5          13.6809    4.9582   15.5016 C.ar      1 STI1        0.0186
      4 C4          14.8459    5.5341   16.0277 C.ar      1 STI1        0.0367
      5 N3          15.9447    4.7940   16.2830 N.ar      1 STI1       -0.2626
      6 C2          15.9678    3.4649   16.0628 C.ar      1 STI1        0.0276
```

The .pdb file would be:

```
ROOT
ATOM      1  C   LIG     1       0.142   -0.047    0.243  0.00
ATOM      2  C   LIG     1       1.012    0.992   -0.126  0.00
ATOM      3  C   LIG     1       2.343    0.913    0.326  0.00
ATOM      4  C   LIG     1       2.785   -0.157    1.111  0.00
ATOM      5  C   LIG     1       1.899   -1.172    1.465  0.00
ATOM      6  C   LIG     1       0.573   -1.115    1.039  0.00
ENDROOT
BRANCH   2   7
ATOM      7  C   LIG     1       0.585    2.177   -0.974  0.00  0.00
ATOM      8  O   LIG     1      -0.462    2.057   -1.689  0.00  0.00
ATOM      9  O   LIG     1       1.325    3.210   -0.925  0.00  0.00
ENDBRANCH   2   7
```

There are several molecular databases which store ligand properties, such as ZINC [16] or PubChem Compound [17]. Apart from the formula and canonical SMILES, they also store other relevant information.

2.3 Comparing Molecules

Molecules can be compared based on their three-dimensional structures. When the molecules are proteins, this process is referred to as "protein structural alignment". When comparing the structures of ligands, it is referred to as "ligand structural alignment" or sometimes "ligand-based virtual screening".

2.3.1 Protein structural alignment

There are a number of powerful protein sequence alignment tools (e.g. BLAST [18]). Aligning sequences can highlight similar regions of 2 (or more) proteins which may show functional, structural, or evolutionary relationships between the proteins. Homology has evolutionary and biological implications (homologous proteins are proteins that have similarity in sequence or structure due to descent from a common ancestor). Two homologous

proteins can have a common function or structure, but sequence similarity does not imply similar function or structure, as two non-homologous proteins may have similar sequences. Furthermore, low sequence similarity does not rule out homology, or common function and structure. The structure of the protein can be used to understand the protein better including its function, mechanisms of action, and structure-function relations. Structural alignment refers to aligning the three-dimensional structure of proteins. It can be a more powerful method for aligning distantly related proteins than sequence alignment.

In structural alignment, the similarity of two three-dimensional objects is assessed. One can imagine superimposing the molecules so that corresponding points are as close together as possible. A common measure of structural similarity is the average distance between these corresponding points. In practice, this is often the Root-Mean-Square Deviation (RMSD) of the corresponding atoms, as shown in Equation 2.1.

$$RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\delta_i^2} \tag{2.1}$$

Where δ_i is the distance between the i^{th} pair of points, and N is the number of points, once the corresponding points have been calculated [19, p. 236].

The remainder of this thesis does not focus on a single structural alignment tool, but as part of the evaluation of the proposed tool-independent framework and methodology (Chapter 8), one tool had to be used. This section explains the choice of DeepAlign. Based on available publications, Hasegawa and Holm [20] estimate that the number of new structural alignment methods has been doubling every 5 years. They provide a review of structural alignment tools, which when extended by the structural alignment tools outlined in [21, 22] makes a list of over 100 distinct structural alignment tools. A subset of 24 stand-alone tools with their respective number of citations (as reported by Google Scholar on 21 July 2016 and 7 April 2018) are shown in Table 2.1.

Table 2.1: Structural alignment tools with citations.

Tool	**Year**	**Cited (2016)**	**Cited (2018)**
DALI [23]	1993	4066	4223
CE (jCE) [24]	1998	2031	2182
TM-Align [25]	2005	1015	1348
MUSTANG [26]	2006	450	558
MAMMOTH [27]	2002	460	509
MultiProt [28]	2004	336	408
FAST [29]	2005	178	196

Continued on next page

Table 2.1 – continued from previous page

Tool	Year	Cited (2016)	Cited (2018)
Matt [30]	2008	160	187
ProBiS [31]	2010	158	173
RCSB PDB - Comparison Tool [32]	2010	93	123
DeepAlign [33]	2013	44	72
SCALI [34]	2005	61	65
LOCK2(FoldMiner [35])	2004	62	64
SA Tableau Search [36]	2010	39	42
CLICK [37]	2011	35	59
SPalign [38]	2012	30	48
TopMatch [39]	2012	32	39
ProSMoS [40]	2007	31	34
MICAN [41]	2013	29	33
CBA [42]	2006	21	29
Smolign [43]	2012	13	16
QP Tableau Search [44]	2009	12	13
SPalignNS [45]	2015	2	4
Fit3D [46]	2015	2	4

Based on the number of citations one can clearly identify a group of older but more cited tools (e.g. DALI, CE, TM-Align, or MAMMOTH), and newer but less cited tools (e.g CLICK, SPalignNS, or DeepAlign).

Barthel *et al.* [47] mention that authors of new tools often use only a small set of test cases to claim benefits of their tool. Common evaluation tests for structural alignment tools use "gold standard" manually curated reference alignments (e.g. HOMSTRAD [48], CDD [49]), classification databases (such as SCOP [50] or CATH [51] - the DALI server no longer provides a database of pre-computed alignments [52, 53]), or a scoring function based on the values such as RMSD [20].

Kim *et al.* [54] evaluated the accuracy of seven tools against the CDD and explain how some programs do not produce high quality individual alignments when measured by geometric match measures. Havrilla and Saçan [55], inspired by the work of Kolodny, Koehl, and Levitt [56], analysed the original set of sequentially diverse protein pairs using another set of structural alignment tools. Their results show that newer structural alignment tools can outperform older ones. Therefore, the classification as new/old can be helpful but not sufficient to choose a structural alignment tool. In this case, a recommendation from

peers was obtained.

The RaptorX structural alignment server [57], which uses DeepAlign [33], was recommended by two biomedical scientists interviewed as part of the thesis. Created in 2013, the number of citations of DeepAlign has nearly doubled between 2016 and 2018 (Table 2.1). The scoring function in DeepAlign calculates a value called DeepScore which represents the equivalence of two residues a_i and b_j from two input proteins (Equation 2.2).

$$DeepScore(i,j) = (max(0, BLOSUM(i,j)) + CLESUM(i,j)) \times d(i,j) \times v(i,j) \quad (2.2)$$

where BLOSUM and CLESUM measure the evolutionary distance of two proteins at the sequence and local substructure levels, respectively. The value $d(i,j)$ measures spatial proximity of two aligned residues once superimposed, while $v(i,j)$ measures hydrogen-bonding similarity [58, p. 144].

An alternative approach would be to use multiple tools, which has been attempted in the past. Barthel et al. [47] have combined several tools to give a consensus similarity profile for a given dataset. To calculate this consensus similarity, it normalises the similarity matrices from the various tools. Kolodny, Koehl, and Levitt [56] propose a "best-of-all" method that uses the best results of six tools. They use four geometric measures to evaluate the quality of each structural alignment.

However, this approach was not used in this thesis. Due to the popularity among peers, which is partly because of the very good performance in the Critical Assessment of protein Structure Prediction (CASP [1]), the tool DeepAlign will instead be considered.

2.3.2 Structural alignment of ligands

Ligands can be compared based on their structure as well. This type of similarity often falls under the category of LBVS. This is because the LBVS methods include molecular similarity comparisons [59]. Some LBVS methods focus on creating "pharmacophores" (models of a hypothetical ligand that binds well), then searching for ligands that are similar to the pharmacophore. This approach assumes that a ligand that binds to a biological target can be described by a set of common features. These features may be, for example: number of hydrogen-bond donors, hydrogen-bond acceptors, and positive or negative charge [60]. Some LBVS methods focus on Quantitative Structure-Activity Relationships (QSARs). The QSAR approaches assume that there is a direct relation between biological activity and molecular structure. According to these approaches, molecules with similar

[1] predictioncenter.org

structure will possess similar biological activities for similar targets, and if the structure is changed there will be a change in the biological activities. Usually, molecules are collected in a trial set and molecular descriptors are calculated. Then, a model is created using a training set. The QSAR model is tested, and based on the correlation to experimental results, it is either reconfigured or accepted for designing novel ligands [61].

The framework and methodology proposed in this thesis do not depend on the choice of tool to assess ligand similarity, but their evaluation (Chapter 8) requires the use of a single tool. Therefore, this section will explain the choice of LIGSIFT as a tool for structure-based comparison of ligands. Two online sources provide a list of LBVS tools. A total of 113 LBVS tools are listed in [62] (53 of which are stand-alone tools), and 35 software tools for LBVS are provided in [63] (number of tools correct as of February 2017).

LIGSIFT [64] is an open-source tool for shape-based alignment of small molecules which is known to perform very well. The following example shows it outperforms other tools. When new tools are developed, they are compared to the existing solutions to show performance benefits. However, in the performance analysis of the recently developed mRAISE [65], the authors were not able to show superior performance over LIGSIFT in a particular performance test. The reference dataset known as Directory of Useful Decoys (DUD [66]) was used to assess whether a ligand for a certain target can be correctly identified within a set of similar decoys. When comparing mRAISE to LIGSIFT, Align-It [67], ROCS [68], SHAEP [69], and MolShaCS [70], the best performance is noted by LIGSIFT.

LIGSIFT measures shape and chemical similarity, and reports the p-value to assess statistical significance of a match between a pair of molecules. LIGSIFT calculates a size-independent score, a version of the widely used Tanimoto Coefficient (TC), which the authors call "scaled Tanimoto Coefficient (sTC)". The sTC is scaled based on a random background distribution (S_0) of shape and chemical TCs, calculated for millions of molecules of different sizes (Equation 2.3). The algorithm behind LIGSIFT uses Gaussian molecular shape overlay in the alignment process [64].

$$sTC = \frac{TC + S_0}{1 + S_0} \tag{2.3}$$

2.4 Molecular Docking and Virtual Screening

The second type of simulation that uses the structure of molecules as input is molecular docking. Molecular docking (often referred to as "docking") can be used to estimate biochemical interactions between two molecules. Particularly important in drug discovery, docking can predict the conformation, pose, and binding affinity of a ligand and receptor

if the 3D structure of both molecules is known. Docking consists of an algorithm to search through the conformational space of the molecules, and a scoring function to estimate the energy between the ligand and the receptor's binding site.

The creators of GOLD describe the docking problem, "the prediction of small molecule binding modes to macromolecules of known three-dimensional structure", to be of "paramount importance in rational drug design" [71].

Docking has been defined as a computational procedure that attempts to predict non-covalent binding of a macromolecule (receptor) and a small molecule (ligand) efficiently, starting with their unbound structures. Its goal is to predict the bound conformations and the binding affinity of the two molecules [72].

Starting from the structures of two unbound molecules, it attempts to predict the structure of the corresponding complex. It predicts "the molecular interaction occurring between drug-like molecules and a therapeutically relevant target" [73]. Docking focuses on finding the low-energy binding modes of a ligand, within the active site of a receptor with a known structure [74]. It aims at the correct placement of a ligand into the binding pocket of a receptor. The binding energy of the resulting complex is then estimated, considering the interactions between ligand and binding site [75]. A ligand that interacts with a receptor associated with a disease, can inhibit its function and act as a drug [76].

Given the atomic coordinates of two molecules, docking predicts their "correct" bound association. Structures of the receptor and ligand in their bound form can be used in a process known as "bound" docking, however, the more difficult predictive "unbound" docking uses the unbound structures to reconstruct a complex. The unbound structure can be native (free in solution in its uncomplexed state), pseudo-native (when complexed with a molecule different from the one used in the docking) or modelled [77].

There are many different docking algorithms, but they all feature these key ingredients: representation of the system, conformational space search, and ranking of potential solutions. Solving the docking problem involves two crucial components - an efficient search procedure, and a good scoring function. Based on the different way to address flexibility in the representation of the system, docking can be classified as:

- Rigid body docking (simplistic model where both molecules are rigid).
- Semi-flexible docking (where one, usually the ligand is considered flexible).
- Flexible docking.

A conformation is the spatial arrangement of atoms in a molecule. The conformational space search can be either a full solution space search, or a gradual guided progression

through solution space. The latter scans only part of the solution space in a random and/or criteria-guided manner, for example using Monte Carlo simulations, simulated annealing, Molecular Dynamics (MD), evolutionary algorithms such as genetic algorithms, or Tabu search. While traversing through the conformational space, the docking algorithm needs to rank the likelihood of particular conformations of the two molecules happening in nature. To achieve this, a docking algorithm uses a scoring function. Examples of the properties that can be considered by a scoring function include: geometric complementarity, intra- and inter-molecular overlap, hydrogen bonds, amino acid and atom-atom contacts, van der Waals interactions, and electrostatics [77].

A docking algorithm defines the aforementioned ingredients, and scientists use the algorithm through a docking program or docking tool. Because docking uses the structure of the receptor, large-scale docking of hundreds of thousands of ligands and one receptor is called structure-based virtual screening (virtual, as opposed to high throughput screening, the automated laboratory experiment). In the remainder of this thesis, VS is used to describe SBVS unless otherwise stated.

2.4.1 Docking tools

The proposed concepts in this thesis are tool-independent, but their evaluation require a particular tool. This section will introduce the selected tools. There are more than 50 docking tools that may be used [63]. Sousa *et al.*, [78] have analysed the number of citations of 22 different molecular docking tools and concluded that AutoDock [79] is the most cited docking tool. DOCK [80] is the second most cited tool when taking only tools that are free for academic use into consideration. After a brief introduction of alternatives, this section will describe AutoDock, its sister-tool AutoDock Vina, and the associated VS tools Raccoon and Raccoon2.

DOCK DOCK was the first and pioneering docking tool [80]. It considered both ligand and protein as rigid. DOCK is still very widely used; the latest series, DOCK 6 includes an improved updated scoring function [81]. DOCK 3 [82] is another actively developed branch, different from the DOCK 6 series.

GOLD Genetic Optimisation for Ligand Docking (GOLD) is one of the early examples of using genetic algorithms in the conformation search phase. It is a proprietary docking tool maintained by the non-profit Cambridge Crystallographic Data Centre [71].

FlexX FlexX [83] is another proprietary docking tool provided by BioSolveIT. In FlexX, the ligand is fragmented into components, then these fragments are docked in the receptor's active site, before the rest of the ligand is incrementally built up.

AutoDock AutoDock (latest version being AutoDock 4.2) [79] is a docking tool that predicts the "optimal bound conformations of ligands to proteins". The AutoDock docking consists of two methods, both of which use approximations [84]:

- Conformational search: The ligand is treated as having flexible torsional degrees of freedom, while bond angles and bond lengths are constant. To perform the search, AutoDock can use four stochastic methods: simulated annealing, genetic algorithms, local search, and a hybrid global-local Lamarkian Genetic Algorithm (LGA).
- Scoring function: interaction energies around the protein are pre-calculated in a "grid map" which is then used as a look-up table. This method treats the protein as rigid, although specific side-chains can be explicitly annotated and treated outside this grid. AutoDock uses a semi-empirical free energy force field to evaluate different conformations [85].

AutoDock is well suited for VS since the grid map needs to be calculated only once, at the beginning [84]. This pre-calculation is done using a separate executable called AutoGrid. Atoms in AutoDock are classified based on atom types such as: non H-bonding Aliphatic Carbon (C), non H-bonding Aromatic Carbon (A), donor 1 H-bond Hydrogen (HD), or acceptor 1 H-bond Nitrogen (NA).

The full list of atom types and other parameters can be viewed in the AutoDock source code [86]. AutoDock uses AutoGrid to calculate interaction energies for each atom type that is part of the ligand and the protein, using a so called "probe" atom and calculating the energy at regular points over a 3D space around the protein. AutoGrid creates a ".map" file for each atom type in the receptor, an ".xyz" file which describes the spatial extent of the grid box, and an ".fld" file which describes the consistent set of atomic affinity grid maps that were calculated together. It also calculates an electrostatics map (.e.map), and a desolvation map (.d.map). In order to do this, AutoGrid requires an input file that specifies the 3D search space around the protein, the types of probe atoms to use, the filename of the protein, and the names of each output grid map. The input file providing this is called a grid parameter file and has the extension ".gpf". These grid maps are used as a lookup table by the docking process to determine the total interaction energy for a ligand and a protein. AutoDock also requires a docking parameter file, ".dpf", which specifies the names of the grid map files and other important parameters such as which conformational search method will be used.

AutoDock has been developed by the Scripps institute. It is provided in a bundle called MGLTools [87] which also contains AutoDock Tools (ADT). ADT is a stand-alone desktop application which provides a Graphical User Interface (GUI) for docking. The .gpf and .dpf input files can be created using ADT, or by running the independent scripts found within MGLTools. Either approach may use a template .gpf file which is used to set the location and extent of the grid maps, or a template .dpf file which has some docking parameters set. More details about AutoDock are provided in [88,89], and on-line [90–92].

AutoDock Vina AutoDock Vina [72] is the newest generation of docking tool developed by the Scripps institute. Partly due to the built-in support for multithreading, AutoDock Vina has a shorter execution time. Chang *et al.* [93] have compared both tools and concluded that the internal changes to the docking algorithm in AutoDock Vina made it more accurate for bigger, more flexible ligands (ligands with more than 8 rotatable bonds). Both AutoDock and AutoDock Vina use the same file format to represent ligands and receptors, .pdbqt. The improvements of AutoDock Vina are mainly in the conformational search and the scoring function with several notable differences [93]:

- Conformational search - It uses the same hybrid global-local search, with a different local optimisation. AutoDock uses small random steps to seek for more favourable conformations, while AutoDock Vina uses a gradient-based optimisation.
- Scoring function - AutoDock Vina has a new differently calibrated scoring function, based on empirically weighed functions and using parameters such as: hydrophobic (van der Waals) interaction, hydrogen bonding, and torsional penalties.

Possibly the most important difference from the user's point of view, is that AutoDock Vina calculates the grid maps automatically without the need to store them in a separate file (i.e. there is no need for .gpf files). Configuration settings can be assigned to a configuration (config) file, usually .conf or .txt. The docking results in AutoDock Vina are clustered and ranked in a more transparent fashion [72, 93].

2.4.2 VS with AutoDock and AutoDock Vina

AutoDock and its sister-tool, AutoDock Vina, are the most popular docking tools for the cohort of interviewees of this thesis (Chapter 4). Both can be used in VS simulations. Scientists can create their own scripts of code, or use other more sophisticated environments. The desktop application that can be used for small-scale docking, ADT, cannot be used for VS. Another desktop application has been developed by the Scripps institute

specifically for VS simulations. There are two versions of this application, Raccoon and Raccoon2. Both will be described in the following paragraphs.

Raccoon Raccoon provides a user friendly GUI for automatic pre-processing and preparation of a VS with AutoDock 4.2. It focuses on "a straightforward data organization important for virtual screening but [does] not provide molecular viewing functionality" [94]. Raccoon automates the creation of ligand files in the AutoDock format, grid map (.gpf) files, and docking parameter files (.dpf).

This desktop application can split multiple-molecule ligand files and filter them using common criteria (such as Lipinski's rules [95], fragment-like "rule of 3" [96], and drug-likeness [97]). The input files are validated ensuring that they have a coherent format and there are no non-standard atom types. Furthermore, Raccoon generates scripts for submission to a Linux cluster with the PBS scheduler, and for post-processing of results. Since Raccoon uses AutoDock 4.2, the grid map files are created once for all atoms in all ligands and proteins that take part of the VS, and they may be reused for each individual molecular docking [84, 98]. The user manual [99] contains more details about Raccoon and its user interface. The Maps tab is used to create the grid maps, generated by AutoGrid. Raccoon provides three different scenarios based on when AutoGrid is going to be executed: "at each job" (for each ligand-protein pair), "now" (once for the protein, all ligand-protein pairs will use it), "never" (the user needs to upload pre-calculated grid maps). Raccoon only prepares the needed files for a VS simulation. The scientist can run the prepared script on the local machine, or on a Linux cluster once they have copied it over. Raccoon does not provide a result analysis GUI, a separate tool called Fox has been developed to provide a GUI for analysing results, which has been subsequently integrated into Raccoon2.

Raccoon2 Raccoon2 [100] is a newer and improved version of Raccoon. It is included in the latest version of MGLTools. The two main improvements in Raccoon2 are the inclusion of analysis features (filtering the results based on several criteria, and visualising the results within the Raccoon2 GUI), and an automatic server connection manager which lets users connect and submit jobs to a cluster directly from the Raccoon2 GUI. However, Raccoon2 does not let scientists conduct the VS on their own computers. Linux clusters with the Portable Batch System (PBS) or Sun Grid Engine (SGE) schedulers are supported. Perhaps most importantly, Raccoon2 uses AutoDock Vina. The ligand and receptor files have to be in the .pdbqt format, but it enables users to create the AutoDock Vina configuration file through its GUI [101].

In some scientific scenarios several tools (e.g. structural alignment, or docking) need to

be executed in one pipeline. Molecular Dynamics (MD) is another type of computational simulation which is sometimes used along with docking to check if the docked ligand-protein complex is stable. MD simulations estimate the movement of the molecules by using Newtonian motion equations to predict the movement of each atom. Due to the large number of atoms, MD simulations have a long execution time to complete, so in practice can simulate very short periods [102]. Chia *et al.* (2010) [103] report that depending on the size of the simulation, MD using GROMACS [104] on a single computer may take days or sometimes weeks to complete. They also show that MD simulations that use complex grid computing infrastructure with 8 processors can simulate less than 3ns of motion per day.

A single docking simulation does not require complex computational resources, but a VS experiment is very computationally demanding, requiring the use of DCIs. One method to execute several tools in a pipeline is to use scientific workflows. DCIs and scientific workflows will be outlined in the following sections.

2.5 Distributed Computing Infrastructures

The underlying computer infrastructure that enables large-scale computationally-demanding execution is known as distributed computing infrastructure or DCI. The main concepts referred to in this thesis are clusters, supercomputers, grid and cloud computing.

High performance computing The computing power of multiple computers (called nodes) can be combined in a High Performance Computing (HPC) cluster. Clusters provide a powerful environment, designed to use parallel computing, which is accessed through a single system image [105]. Efforts to combine several computers started in the late 1960s when terms such as Cluster Of Workstations or Network Of Workstations were used. Clusters that do not require specialised components, but use commodity hardware have been popular since the publication of the Beowulf cluster architecture in 1995 [106]. Nowadays, most commercially used HPC clusters would be made up of purpose-built hardware. An HPC cluster requires a job scheduler, some of the most common ones include the Slurm Workload Manager [107], descendants of the PBS [108] such as PBS Professional [109], or descendants of the SGE [110] such as Open Grid Scheduler [111].

Supercomputer is a term used for specialised HPC computers. Traditionally a supercomputer is a single machine with powerful processors capable of parallel processing. In the early days of clusters, data communication between cluster nodes was noticeably slower than communication within a single machine. Today's clusters use high-speed network

technologies for communication between nodes, and the term supercomputers could be used for large HPC clusters. For instance, almost 90% of the TOP500 list, originally created to list the world's fastest supercomputers, uses a cluster architecture. Fifteen years ago, this number was just over 16% (not including "constellation" clusters where there are more processors per node than there are nodes) [112].

Grid computing Both single-machine supercomputers and clusters are computers that are based at a single location, known as non-distributed computers. Conversely, a computer grid is by definition made of geographically-distributed nodes. The term "grid computing" was coined in the mid 1990s to describe technologies that would enable the use of computing power on demand. Inspired by the concept of utility computing which was first described in the mid 1960s, researchers envisaged that standardising protocols used to request and serve computing power would create a computing grid analogous to the electric power grid. Computer engineers created implementations of grid computing environments (e.g. EGEE, TeraGrid, Open Science Grid), but no viable commercial grid computing provider emerged. This enabled the advent of cloud computing in the late 2000s [113].

Cloud computing Cloud computing is a paradigm based on virtualisation, which "enables ubiquitous convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction" [114]. It traces its origins to the concept of utility computing, where computing was envisaged to become a public utility such as the land-line telephone system, first discussed in the early 1960s. Cloud computing has evolved out of grid computing, as a result of the shift of focus from the storage and compute infrastructure to an economy-based infrastructure that delivers computing resources and services [113]. Indeed, with cloud computing one can easily rent computing resources and pay by the usage. Other characteristics include [115]: multi-tenancy and resource pooling, on-demand usage (automated self-provision of computing resources), ubiquitous access, and elasticity (transparent scaling of resources in line with run-time requirements).

There are three common cloud delivery models and four common cloud deployment models, as outlined in [115]. A cloud delivery model is the specific combination of resources offered by cloud providers, such as:

1. Infrastructure-as-a-Service (IaaS): "raw" resources and detailed configuration.
2. Platform-as-a-Service (PaaS): a pre-configured "ready-to-use" environment.
3. Software-as-a-Service (SaaS): the use of a cloud-deployed software product.

A cloud deployment model is the specific type of cloud environment offered, such as:

1. Public clouds: publicly accessible, owned by a third-party.
2. Community clouds: accessible and perhaps owned by a particular community.
3. Private clouds: owned and accessible by a single organization.
4. Hybrid clouds: an environment composed of 2 or more deployment models.

The elasticity and scalability, both through horizontal scaling out or in (allocating or releasing resources of the same type) and vertical scaling up or down (increasing or deceasing the capacity of the currently used resource), are very beneficial for VS. Cloud computing can be used efficiently for small as well as large VS simulations. The on-demand and measured usage can make VS simulations more accessible for biomedical scientists around the world, lowering the cost of using the required DCI. Furthermore, if VS is implemented based on the SaaS delivery model, biomedical scientists will always have access to the latest version of the simulation software. Scientists and students without access to expensive DCIs, and without experience in configuring them, will be able to run a VS easily.

2.6 Existing Virtual Screening Applications that Use Cloud Computing

Cluster or grid computing resources have been common for VS experiments [75, 116–119]. Applying cloud computing for such experiments is still relatively new with much lower number of examples.

De Paris *et al.* [120] have developed wFReDoW (acronym for "web Flexible Receptor Docking Workflow"), a web-based environment for docking fully flexible receptors using AutoDock 4.2 as the docking engine. They model the flexibility of the receptor using snapshots of MD simulations (using the SANDER module of AMBER [121]). They use ligand structures from ZINC and 3100 conformations of a receptor generated by MD simulations. They have set up a virtual HPC environment on the commercial Amazon Elastic Compute Cloud (EC2). It is a Message Passing Interface (MPI) environment containing 5 high-CPU extra-large "c1.xlarge" Amazon EC2 instances, each equipped with 8 cores with 2.5 EC2 computer units, 7 GB of RAM, and 1,690 GB of local instance storage (one EC2 computer unit corresponds to CPU capacity of 1.0 - 1.2 GHz 2007 Opteron or 2007 Xeon processor).

Ellingson and Baudry [122] have used AutoDock 4 in AutoDockCloud, an environment based on Hadoop [123] on a private cloud. According to them, high-throughput virtual docking on a cloud architecture has many potential advantages, such as: providing an

efficient and well-validated VS technology to laboratories and classrooms that do not have computational wealth or expertise to overcome challenges, and providing it as SaaS, enabling researchers to always have access to the most updated versions without having to reinstall software. AutoDockCloud uses Kandinsky, a private cloud at the Oak Ridge National Laboratory, with 57 reserved 16-core nodes, enabling 570 simultaneous docking runs. They have used 2637 ligands (67 active and 2570 decoys) from the DUD and the human oestrogen receptor alpha agonist protein (pdb id: 1L2I). They have used ADT to create .pdbqt, .gpf and .dpf files. They conclude that AutoDockCloud does not affect the biochemical results and has finished the docking runs 450 times faster than a non-parallel execution would. However, it only handles the docking stage and not the pre- or post-docking. A big challenge of automating this is parsing different input files (.mol2, .pdb, and .sdf). With regards to post-docking, they envisage extracting the best outputs in reduce tasks in a future version.

Kiss *et al.* [124] have ported AutoDock and AutoDock Vina on the VENUS-C Windows Azure-based cloud computing service. Their implementation includes an administration, deployment, and end-user component. It enables scientists to submit, monitor and retrieve results of a VS. They use a desktop application bundle that scientists need to install to their own computer in order to remotely manage the experiments. They have conducted a VS using a library of 10,000 ligands and a protein (generated from a short MD run on the initial structure) on 20 "extra small" Azure instances (a single 1GHz CPU core with 768 MB RAM and 20 GB storage). During their tests, more than 40,000 docking simulations have been done and more than 110,000 CPU hours have been used.

2.7 Scientific Workflows

Scientific scenarios that require several tools to be executed in one pipeline often use scientific workflows. A typical VS simulation requires pre-docking, core docking, and post-docking steps. A pre-docking step can be formatting input files, the core docking can include multiple steps and tools e.g. AutoDock 4.2 requires running AutoGrid followed by AutoDock, and a post-docking step can be inspecting the predicted complex by the user. It is worth noting that the desktop application Raccoon2 includes code for the mentioned pre- and post-docking steps, while it has hard-coded commands that will submit docking jobs to an HPC cluster. Instead of hard-coding such methods, using a scientific workflow provides an interoperable solution that works on various DCIs. A scientific workflow is a pipeline made up of the steps required for a computational experiment. It is a network (or graph) of independent analysis items that can be, for example: database access, calculation, data analysis, or data visualisation. Scientific workflow management

Table 2.2: Basic comparison of workflow engines.

Workflow Engine	Cloud	Grid	HPC	GUI	Docking	Local Support
Kepler	✓	✓	✓	✓	✓	×
Nextflow	✓	✓	✓	×	✓	×
Taverna	✓	✓	✓	✓	✓	×
WS-PGRADE	✓	✓	✓	✓	✓	✓

systems use workflow engines and provide a convenient way to represent and develop complex applications composed of multiple steps and executables. In some cases, science gateways are developed, summarising multiple workflows in one portal and enabling the use of complex DCIs with little or no expertise required. Importantly, scientific workflow engines inherently parallelise the task, and include built-in plug-ins to various types of DCIs thus eliminating the need for the workflow developer to hard-code commands that run on a DCI.

Workflow engines that have been used for bioinformatics include Kepler [125], Nextflow [126], Taverna [127], WS-PGRADE/gUSE [128] and many more. Table 2.2 shows that all aforementioned workflow engines support execution on clouds, grids, or HPC clusters. This interoperability is crucial, since it provides the ability to utilise the same workflow on different computing infrastructures. Nextflow is a workflow engine that is popular in bioinformatics, but does not include a graphical method for defining workflows. Less tech-savvy biomedical scientists would appreciate a GUI used for creating and understanding workflows. All four workflow engines have been used in existing solutions for docking simulations. WS-PGRADE/gUSE has been the workflow system of choice at the University of Westminster for more than a decade. Researchers at the UoW, in close collaboration with the SZTAKI institute in Hungary, represent the leading experts in WS-PGRADE workflow development, making WS-PGRADE stand out due to the local support available. Therefore, WS-PGRADE was chosen as a workflow engine used in the implementations provided in this thesis. Please note that the concepts described in this thesis do not depend on the choice of scientific workflow system.

2.7.1 WS-PGRADE/gUSE

The grid User Support Environment (gUSE) is a back-end service stack for creating science gateways that execute applications on various DCIs. Providing well-defined services for realising the workflow management back-end of the WS-PGRADE portal is one of its main functionalities. The workflow-centric generic and open-source DCI gateway framework known as WS-PGRADE/gUSE is the combination of a WS-PGRADE portal, WS-PGRADE workflows and gUSE services. A science gateway developed using this

framework is often referred to as a WS-PGRADE/gUSE gateway. Originally supporting the needs of application development for grid computing, today WS-PGRADE/gUSE also supports developing applications for parallel execution on clouds. It includes a component called DCI Bridge, which provides uniform support with a well-defined communication interface to access many different DCIs [129–131].

A P-GRADE portal (short for Parallel Grid Runtime and Application Development Environment) was a general-purpose e-science portal for development of grid applications. It was the front end of an environment for running workflows on a DCI, while the back-end consisted of a P-GRADE workflow manager and a grid middleware. P-GRADE is the name of a deprecated version. The new version which provides many advanced features is known as WS-PGRADE. It originally stood for "Web Services - Parallel Grid Runtime and Developer Environment", but now the abbreviation WS-PGRADE is used as an orphan initialism. The WS-PGRADE portal is the default user interface of gUSE. It is a web portal based on Liferay [132]. It allows scientists to run pre-configured WS-PGRADE workflows on various DCIs from their web browsers. In a similar manner it enables workflow developers to develop workflows through the portal. Some provenance information can be found in the WS-PGRADE portal, but this is only available for the user's own workflows. A scientist can see their previous runs with dates, and download workflow configuration, input, and output files. The scientist's own workflows are reproducible by downloading and then using the downloaded workflow files [128, 131, 133–135].

P-GRADE workflows were data-flow directed acyclic graphs where nodes represented execution blocks which had input and output ports and could be executed in parallel. WS-PGRADE workflows are an extended version of P-GRADE workflows, and have their own XML-based workflow language along with many new capabilities. P-GRADE workflows were "concrete" workflows where the workflow nodes were simply executed in parallel. Parameter sweep operations, where a set of inputs is provided and the node is executed as many times as the number of inputs, were possible only by using special kinds of nodes (generator and collector nodes). WS-PGRADE workflows do not have these restrictions and apart from concrete workflows, WS-PGRADE also supports new concepts such as abstract workflows, workflow instances, and templates [128, 131].

Simulation applications that should be executed repeatedly with many different input sets are known as "parameter sweep" applications. A WS-PGRADE workflow node can have a parametric input port associated with a set of input files. Nodes that have at least one parametric input port are called "parametric" nodes. If the parametric node has one parametric input port the node will be executed once for each input file associated with it. Parametric input ports, which are not connected to a generator node, expect to receive an archive following a specific naming convention. The input file must be called

"ParamInput.zip" and it must contain a set of input files named exactly: "0", "1", "2", etc. Furthermore, the number of files that are associated with the parametric input port must be specified beforehand in the port's configuration. When a node has more than one parametric input port, it may use a dot-product or a cross-product method to combine the input files. A dot-product executes the node using the first input file of all parametric input ports, then using the second input file of all parametric input ports, then the third, and so on. In contrast, the cross-product uses a Cartesian product combination of all the input files among the parametric input nodes [131, 135].

The gUSE RemoteAPI There are three ways to run simulations on DCIs using the WS-PGRADE/gUSE framework: an application-specific user interface can access the DCI Bridge through an OGF BES job submission interface, a customised portal can use the Application Specific Module (ASM) API, or an existing application with a GUI which is not a portal or even without a GUI can access gUSE services directly through the RemoteAPI. The RemoteAPI is an Application Programming Interface (API) that allows remote submission and management of a WS-PGRADE workflow. In other words, using a WS-PGRADE workflow from within a code segment which is not part the WS-PGRADE portal or gUSE. [131]

The RemoteAPI can be used to adapt an existing user environment. The ASM API mainly supports the development of portlets or other GUIs within a portal, while the RemoteAPI is designed for direct access to gUSE services. The ASM API is a Java-based API, while the RemoteAPI is used via HTTP regardless of the programming language. During workflow submission, the RemoteAPI does not require user registration to a WS-PGRADE portal, but it does require a valid well-parametrised WS-PGRADE workflow. It creates a new temporary user and submits the workflow on behalf of this user. Methods for checking the status of the workflow and downloading the outputs of the execution are provided. Once downloaded, the output files and all information about the temporary user are deleted from the server [131]. Examples using the RemoteAPI include the agINFRA EU project, where an agricultural research community has used existing tools to access the services of WS-PGRADE/gUSE through the RemoteAPI [136]. Prerequisites for using the RemoteAPI include: a gUSE server with enabled RemoteAPI, RemoteAPI credentials, and a client that will send well-parametrised description of the WS-PGRADE workflow. More details about WS-PGRADE/gUSE and the RemoteAPI can be found in [137–139].

2.8 Science Gateways and Workflow Repositories

This section will provide several examples of systems that use various scientific workflows, known as science gateways. Science gateways provide a single point of access to multiple workflows in the same or related domains. Examples of a WS-PGRADE-based science gateways in the domain of molecular simulations, but also in non-biological domains will be mentioned. The notion of provenance in these systems will be touched upon as well as examples of repositories that store workflows. To ensure that the results of any computation are trustworthy and reproducible, it is important to keep track of all the steps taken to produce those results. This type of information is known as provenance. According to the general definition of provenance provided by the PROV standard, provenance is information about entities, activities, and people involved in producing a "piece of data or thing", which can be used to form assessments about its quality, reliability or trustworthiness [140]. The examples provided can be the starting point for the implementation of a system that conducts docking simulations and stores the simulation results. However, the repositories shown here store the workflow description and not necessarily the results of an execution of a particular workflow.

CloudSME The Cloud Computing for Simulation in Manufacturing and Engineering (CloudSME [129]) was a European FP7 project which investigated how cloud computing can be used for Modelling & Simulation (M & S) in manufacturing and engineering. It also promoted cloud resources and enabled wider use of simulation technologies in Small and Medium sized Enterprises (SMEs) in the domain of manufacturing and engineering. In order to do this, CloudSME provided M & S as a Service using their own CloudSME Simulation Platform. The CloudSME Simulation Platform is based on a WS-PGRADE portal, WS-PGRADE workflows, and gUSE services (including the DCI Bridge) which are integrated with the CloudBroker Platform.

The CloudBroker Platform [129] is a cloud computing middleware and an application store developed by CloudBroker GmbH. It provides a web interface which can be used to deploy and execute an application in a cloud, and monitor its behaviour. The CloudBroker Platform is connected to various kinds of clouds, including commercial (e.g. CloudSigma, Amazon Web Services) and open-source (e.g. OpenNebula, OpenStack). Within CloudSME scientists can run simulations on a cloud directly from a WS-PGRADE portal. The CloudBroker Platform provides its own API. For simulation software that requires simple hosting on a cloud, using the CloudBroker APIs directly may be sufficient. However, it has been noted that for simulations that require high performance computing, using a WS-PGRADE/gUSE framework and its APIs is preferred [129].

MoSGrid The Molecular Simulation Grid (MoSGrid) is a WS-PGRADE/gUSE-based environment specifically designed for scientists from three domains of molecular simulations: molecular dynamics (often abbreviated to MD), molecular docking, and Quantum Chemistry (QC). It uses the Molecular Simulation Markup Language (MSML [143]), a description language which describes the molecules and results from all three types of simulations. MoSGrid enables a scientist to archive the results of their calculations in a repository and share them with another user. Users can view or download the results of a simulation (for MD simulations, even the intermediate results for an ongoing simulation). MSML enables linking information from different simulations, in the case of docking, for instance, one may dock the same ligand library into two different target proteins. The MSML files are indexed and searchable using Apache Lucene [144]. They contain information about properties of a workflow task (e.g. nodes, cores, or memory), input data (the molecular structure), as well as the resulting output files [143, 145].

MSML is a derivative of the XML dialect used to describe chemical molecules and processes called Chemical Markup Language (CML). Every MSML file is a valid CML file, using only what is necessary for MD, docking, and QC simulations [143, 146]. MSML uses the Computational Chemistry CML convention without any modifications, and three custom-made CML dictionaries including a dictionary which defines all the steps of the docking workflow [147]. The open-source tools of CADDSuite have been used for pre-processing and post-processing the molecular structures. CADDSuite, FlexX [148], or AutoDock [79] can be used for the main docking step [143, 146].

SHIWA repository The SHIWA Simulation Platform (SSP) [149] is an implementation of a workflow interoperability concept. An abstract workflow, which defines the workflow formally without specifying a workflow engine, is introduced and stored in SSP. The abstract workflow can then be executed using a number of workflow engines. SSP contains a workflow repository, submission service, proxy server, and a portal (the portal is composed of gUSE services as back-end and a WS-PGRADE portal as front-end).

Based on the description of workflows, the workflow repository, known as the SHIWA Repository, manages workflows and workflow engines. It provides access to three types of users: e-scientists who can browse or search the repository for a workflow they can then submit through the submission service; workflow and workflow engine developers, who can describe, update, or delete workflows and workflow engines; and repository administrators, who manage and maintain the repository [149].

Semantic Provenance Processor A command-line tool, the Semantic Provenance Processor (SPP) [150] uses Taverna workflows and stores their provenancce in the Janus

[151] format, allowing input/output files and other annotations to be exported into the myExperiment repository [152] with example data. SPP pre-dates the development of the "taverna-PROV plug-in" [153] and its integration into the Taverna environment. It uses an RDF triple store called 4store [154] and SPARQL [155] to query it.

Ouzo / ProQA The Ouzo [156] semantic web uses semantic annotations to combine meta-data about Taverna workflows, such as inputs and outputs used, with other types of provenance data provided. It can be queried using the Provenance Query and Answer (ProQA). Within Ouzo, a data store called "Baclava" and a provenance store called "KAVE" have been defined. Relationships between the different types of data within a workflow, and additional information such as the creator of the workflow can be analysed.

BioWEP BioWEP [157] is a portal that lets users run Taverna or BioWMS [158] workflows. Workflows have been predefined and created by administrators, but the user can upload their own workflow and, if approved, run it. BioWEP stores inputs and outputs in a database, as well as intermediate files. Users can search for workflows based on the type of input/output that have been used, but the user can only view their own workflows.

myExperiment The myExperiment [152] repository is a repository of Taverna workflows. It only includes the workflow definitions and additional data such as the creator of the workflow. It does not include any data about workflow executions.

Kepler repository The Kepler Analytical Component Repository (KACR [159]) is a repository for workflow definitions. It stores workflow definitions and allows users to download a ".kar" file. This type of file is an archive of XML documents describing the workflow, but it lacks any information about input/output files.

2.9 Frameworks Used in Bioinformatics

Using a framework instead of creating everything from scratch can produce a more accurate, less error-prone, easy to maintain complex software system. There are many definitions of a framework. A pre-made library containing classes and methods that should be used within the code, can be considered a framework in software engineering. These types of frameworks are sometimes referred to as APIs or application frameworks. They may be considered a group of several tools developed by the same team. According to

this definition, there are many frameworks used in structural bioinformatics, such as the open-source: BALL [160], Biopython [161], or CDK [162].

BALL BALL is an application framework developed since 1996 at the University of Tübingen, written in C++, specifically designed for software prototyping and Rapid Application Development in computational molecular biology and molecular modelling. It contains methods used in docking, for instance: adding H atoms, or energy evaluation (force fields). It has been extended with the BALLView visualizer and integrated in the Galaxy workflow management system as Balaxy [160, 163, 164].

BioPython BioPython is a set of Python libraries (or APIs) for bioinformatics problems which has been started in 1999. Examples of the methods it provides include: reading/writing sequence file formats, 3D structures, and interacting with other tools. It is just one element of the "Open Bioinformatics Foundation" which includes: BioPerl, BioRuby, BioJava, and BioSQL [161, 165].

CDK The Chemistry Development Kit (CDK) is a Java library for structural Chemo- and Bioinformatics. Examples of the methods provided in CDK include methods for common tasks such as 2D and 3D rendering of chemical structures, and input/output routines. It has been integrated with the Taverna workflow management system as CDK-Taverna [162, 166, 167].

On the other hand, the definition of a framework used throughout this PhD thesis is more conceptual and focuses on clearly describing all elements and interfaces that constitute a large system. It is independent of the implementation or the programming language of choice. It is generic because it is not tailored to a specific scenario, however it is meant specifically for creating software systems that use molecular docking results.

The use of formal methods based on mathematics can improve the quality of software by producing precise, unambiguous documentation of the software system, where the information is structured at an appropriate abstraction level. The formal documentation of a system can be used to support its design, development and maintenance [168]. This is why formal methods have been chosen to describe the framework in this thesis. The framework provides a formal description of all the element types and interfaces.

Z notation Z [169, 170] (often called Z notation) is a state-based formal notation based on the Zermelo-Fraenkel theory. It allows for the grouping of formal rules in so-called "schemas" that contain logical and discrete-mathematics expressions describing part of a

system. A schema describes a state and can contain state variables. Operations upon the state use mathematical conventions and can be defined on elements, for example on sets, tuples, relations, or functions. Z can be used to model an abstract formal specification of the behaviour of a system [171], or to formally describe: workflows and meta-workflows [172, 173], federated clouds [174], or even the behaviour of a cell [175]. One of the most successful projects using Z, as shown in [168], is a software system known as the Customer Information Control System. Its development has began in the 1970s, and by 1980 it had so many extensions that a redesign was needed. As part of this redesign, the mathematical Z notation was used. Woodcock and Davies [168] report that even programmers with no previous experience in mathematics found Z easy to learn and to apply. One of the key aspects they mention, the use of natural language in Z, will be used in the formal description presented in this thesis. Namely, the mathematics can be related to objects in the real world through judicious naming of variables or additional textual commentary. Z notation is ideal for describing the framework because of its understandable syntax and its versatility. Z or languages based on Z have been used to describe systems and their properties in [176, 177].

2.10 Conclusion

The existing research areas that are important for any system that uses bioinformatics tools have been described in this chapter. Two types of bioinformatics tools, structural alignment and molecular docking tools, have been introduced. In order to explore how they help scientists find out how structurally similar two molecules are or where and how two molecules bind, the description of molecular structures which they use as input, as well as existing algorithms have been overviewed. Since running a large number of these simulations is computationally demanding, solutions such as distributed computing infrastructures and scientific workflows have been developed. Unlike existing frameworks that are used in bioinformatics, the framework proposed in this thesis is a higher-level conceptual framework. It is tool-independent, however for its evaluation a particular tool had to be chosen for the respective element. Some of the choices made were outlined in this chapter.

Chapter 3

Extension of Desktop Applications with Cloud Computing Capabilities

3.1 Introduction

Large-scale docking, known as VS, is often used in drug discovery. Although a single docking simulation is relatively short, a VS experiment is computationally demanding, requiring the use of complex DCIs. This is why user-friendly domain-specific web or desktop applications that enable running simulations on powerful computing infrastructures have been created. Cloud computing provides on-demand availability, pay-per-use pricing, and great scalability which can improve the performance and efficiency of scientific applications. Using cloud computing for biomedical projects has advantages and disadvantages, as noted in [178]. Cloud computing could decrease the cost of running a VS simulation by minimising direct costs such as hardware purchase costs, network services, or electricity. This can make VS simulations more accessible, particularly for scientists without access to computing clusters or other expensive DCIs.

Scientific workflow systems provide a convenient way to represent and develop complex applications composed of multiple steps and executables. In some cases, science gateways are developed, providing a user-friendly way to run workflows. There are several examples of science gateways that use workflows to run VS simulations [75, 116, 179]. However, all of these solutions require life scientists to become familiar with new, typically web-based user interfaces, and significantly restrict the use of the docking software for the sake of simplicity and ease of use. On the other hand, there are popular desktop applications for VS simulations which offer greater flexibility, such as Raccoon2 . Unfortunately, these desktop applications are either restricted to local resources, or require expensive compute clusters and significant IT support to run them on DCIs. These tools typically cannot

utilise cloud computing resources. This is relevant for other domain-specific desktop applications, not only for VS simulations. For instance, in the wider field of bioinformatics many popular desktop applications exist, including several recent examples [180–182].

This chapter investigates how domain-specific desktop applications can be extended to run scientific simulations on various clouds. It proposes a generic approach to extend domain-specific desktop applications to execute workflows on clouds, while retaining the same familiar GUI presented to end-users. A proof of concept is implemented using the VS desktop application Raccoon2, WS-PGRADE workflows, and gUSE services with the CloudBroker platform. The presented analysis illustrates that this approach of extending a domain-specific desktop application can run workflows on different types of clouds, and indeed makes use of the scalability provided by cloud computing. It also facilitates the execution of virtual screening simulations by biomedical scientists without requiring them to abandon their favourite desktop environment and providing them resources without major capital investment. The work presented in this chapter has been published as a conference paper [12] and forms a part of a journal article currently under review.

3.2 Generic Concept to Add Cloud Computing Capabilities to Desktop Applications

The aim of the generic concept is to enable existing desktop applications to access heterogeneous cloud computing resources. This should be achieved without major reengineering of the desktop application and without further burdening the end-user. Ideally, end-users should be able to design and execute the experiments in the exact same way they have done earlier, now with the possibility to send the computations to cloud computing resources. In order to achieve this, the desktop application should use a set of services (Cloud Access Services - CAS). CAS should be available from an API in order to facilitate its integration to the GUI of the desktop application. Additionally, CAS should provide access to a wide range of cloud computing resources, and enable the design and execution of complex application scenarios, such as parameter sweep workflow applications, typically used in scientific computing.

The integration requires two major steps from a developer, as illustrated in Figure 3.1. During the first step, CAS is configured to run the application in the cloud. This step typically requires preparing workflow applications describing the experiment, and configuring CAS to interface with the desired cloud resources. In the second step minor modification of the GUI of the desktop application is required, as well as integrating the submission of the workflow and retrieval of the results within the application's back-end.

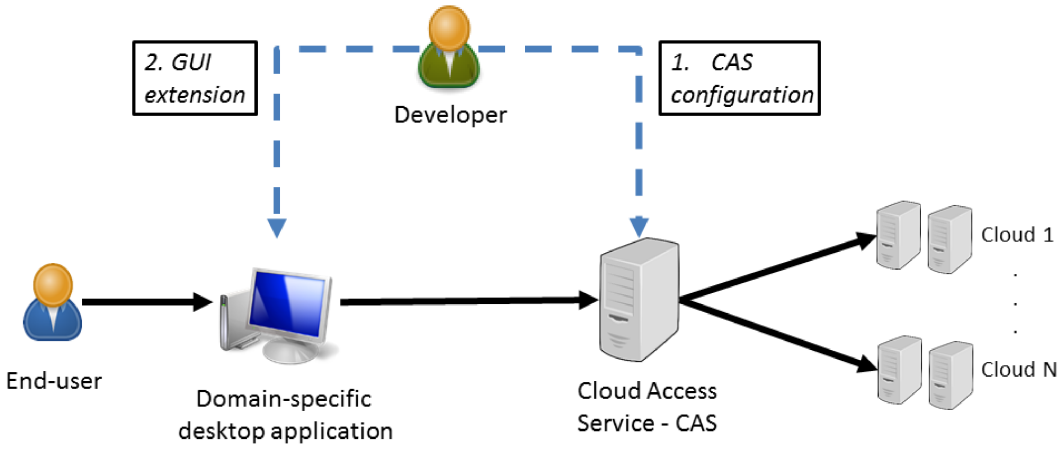

Figure 3.1: Generic concept for extending desktop applications to run on clouds.

Instead of implementing CAS, the core component of this conceptual architecture from scratch, existing tools to support the creation of parameter sweep workflows and interfacing with cloud computing resources can be applied. This approach speeds up the development and has the potential to result in a mature and highly reliable solution. The rest of this chapter describes this approach using a set of existing services and components as the selected CAS and their integration to a VS desktop application.

3.3 Reference Implementation: Extension of Raccoon2

A reference implementation of the proposed concept has been completed. When implementing the generic concept of Figure 3.1, the domain-specific desktop application is Raccoon2; the CAS is composed of a gUSE server connected to the CloudBroker Platform, a WS-PGRADE portal for workflow development, and the CloudBroker web interface for deployment; while the cloud infrastructures are the UoW OpenStack cloud, and the CloudSigma cloud (Figure 3.2). Please refer to Chapter 2 for a detailed description of the theoretical background and specific tools used in this implementation. The components of the CAS used in this solution are existing tools, the main coding of this implementation consists of extending the code of the desktop application Raccoon2.

As described above, the development is divided into two major steps: configuration of the CAS (1) and modification of the desktop GUI (2). First, the CAS is prepared to execute the VS experiment which includes creating the required WS-PGRADE workflow. When accessing gUSE through the RemoteAPI, a valid well-configured WS-PGRADE workflow needs to be attached. To simplify this step, a developer can create the workflow using a WS-PGRADE portal, test it with test input data, and then export it. The exported

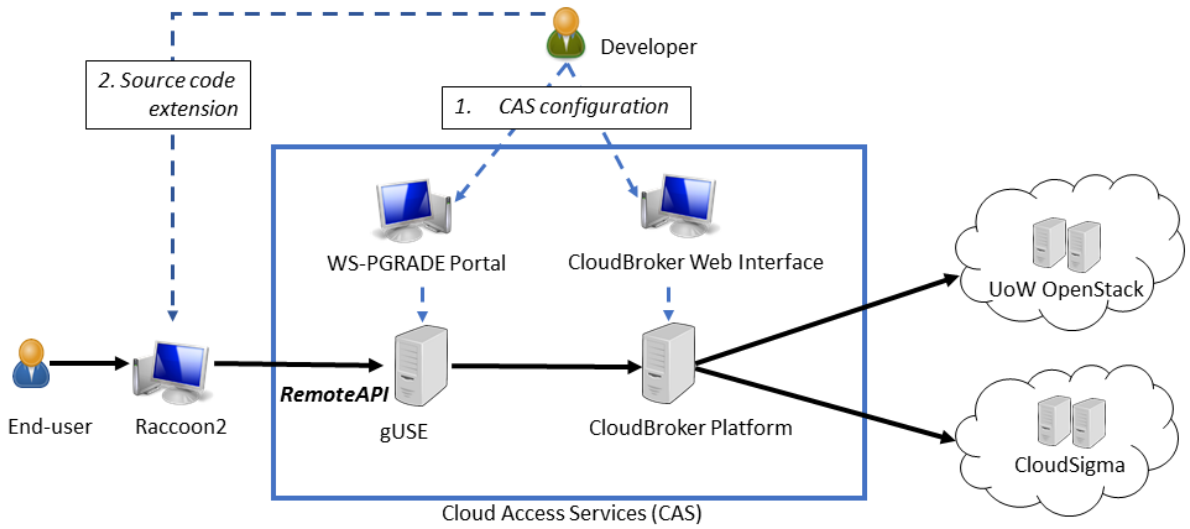

Figure 3.2: Architecture of the reference implementation using Raccoon2, WS-PGRADE/gUSE, CloudBroker, and the UoW or CloudSigma clouds.

workflow can be configured from the code of the domain-specific desktop application and attached to a RemoteAPI call, rather than created from scratch. To conclude (1), the executable files that are needed to run the workflow should be deployed to the cloud, using the CloudBroker platform. In step (2) the source code of the domain-specific desktop application is extended, in order to add an option to the GUI to execute the simulation on a cloud and to make the appropriate RemoteAPI calls in the back-end. The next paragraphs will elaborate on these steps.

Motivation for extending Raccoon2 The latest version of Raccoon2 [101] can be only used to deploy AutoDock Vina and run VS simulations on a Linux PBS or SGE. After obtaining feedback from life scientists it became evident that this is not ideal. The domain scientists lack the required computational expertise and may need additional training to use HPC clusters. The purchase and maintenance of a computing cluster is very costly and at the time this project started, the UoW did not have a functioning cluster. There was an attempt by a scientist to use a "virtual" PBS cluster set up on the UoW cloud, however this was not successful. Many scientists around the world face similar issues, as they do not have access to a cluster. This is a barrier to running VS simulations with Raccoon2. Updating Raccoon2 with cloud computing capabilities is one solution to this problem. Note that the same concept can be implemented with another tool instead of Raccoon2.

3.3.1 Step 1: Configuration of the CAS

Creating a WS-PGRADE workflow The execution steps of the domain-specific desktop application are recreated using a WS-PGRADE workflow. In this particular case a simple one-node workflow with four input (ligand files, receptor file, Vina configuration file, and an additional file to overcome an output names issue) and one output (the zipped results from the multiple docking runs) ports were created. In an optimised version, the last input port would not be necessary. Please note that based on the domain-specific desktop application, more complex workflows may be required. For instance, Raccoon (Figure 3.4) requires a multi-node workfow due to the multiple steps needed by AutoDock 4.2. However, Raccoon2 uses AutoDock Vina which is made up of one major steps, and all the pre- and post-docking steps are completed outside of the workflow. This workflow was created using a WS-PGRADE portal where it was tested, and then exported. Once submitted, the workflow invokes CloudBroker's execution script which runs AutoDock Vina for each ligand it receives as input. In order to run this workflow on many cloud instances, the extended code of Raccoon2 splits the set of ligands into as many zip archives as the number of instances, and submits a separate workflow to each instance.

Figure 3.3: WS-PGRADE workflow for the Raccoon2 extension.

Configuring CloudBroker The CloudBroker deployment process requires: an application deployment script, and an application execution script with an optional application bundle. An image of the operating system (OS) needs to be installed in the image repository of the target cloud. Then, the required dependencies need to be installed using a deployment script. The deployment script is run only once and it is used to prepare the OS image. A snapshot of the prepared image is then created and it is used for future jobs. The execution script needs to validate and manage the inputs, execute the application and return the outputs to a particular folder. Using the CloudBroker Platform web interface, the deployment can be prepared, generated and then activated [183].

3.3.2 Step 2: Modification of the Raccoon2 GUI and back-end

In order to conduct VS simulations on a cloud, the WS-PGRADE workflow should be submitted using the gUSE RemoteAPI. A WS-PGRADE workflow consists of an XML file (workflow.xml) which describes the workflow and the input files. The XML file contains other valuable information, such as which kinds of cloud instances would be used. To fill in the cloud configuration information correctly, the Raccoon2 source code was extended with a section which enables users to select the number of cloud instances, their size, the name of the cloud, and the region.

Before submitting the workflow, the WS-PGRADE workflow XML file is updated to include the cloud configuration data selected in the GUI by the scientist. The same extended GUI is valid for any cloud provider supported by the workflow management system, but if another DCI is used instead of clouds this would need to be modified. This modification of the GUI is minor as it requires a small amount of changes in the code while remaining seamlessly integrated with the original GUI of the desktop application. Within the original Raccoon2 GUI, the user can attach a set of ligands and a receptor - this remains unchanged.

The updated workflow.xml file is then archived along with the rest of the input files, following the WS-PGRADE naming convention. Apart from the attached workflow, the RemoteAPI methods require authentication. Namely, a RemoteAPI password (set by the gUSE server administrator), and CloudBroker user credentials (username and password) are required. The scientist can enter this information using the extended GUI of Raccoon2. Finally, Raccoon2 can submit the workflow by calling a gUSE RemoteAPI method by sending an HTTP request using the Python module "Requests". The RemoteAPI method returns a workflowID, which is used to check the workflow's status. Monitoring the VS simulation is done by polling for the status of the workflows using the RemoteAPI (in the current implementation, every 20 seconds). The status is displayed on the GUI and if there were errors the workflow can be resubmitted. Once a workflow has finished, a final RemoteAPI call retrieves the output.

When all the workflows complete successfully, their outputs are downloaded as ZIP archives and only the relevant AutoDock Vina result files are extracted into a result folder. This result folder can be opened directly from the Raccoon2 GUI as part of the result analysis and visualisation panels. These panels remain unchanged, and the filtering and visualisation features can be used, exactly as in the original Raccoon2. The extension of the original Raccoon2 was written in Python [184].

3.4 Additional Implementation: Extension of Raccoon

Independently from the extension of Raccoon2, the same generic concept was used to extend Raccoon. It shows how the implementation can describe multiple workflows and execute the correct one based on the user's selection. The main difference between the two VS desktop applications is that Raccoon uses the docking tool AutoDock 4.2, while Raccoon2 uses AutoDock Vina.

In Raccoon, the user can specify when the AutoGrid component is executed (AutoGrid generates interaction maps needed by AutoDock). This can be: "run AutoGrid on each job", "now (and cache the maps)", or "never (maps are already calculated)". In the original Raccoon a different script is executed based on which of these three options the user selects. In the extended version using the generic concept, a different workflow is executed based on this selection. In the first case the AutoGrid component requires the GPF and DPF input files to be prepared, which is the responsibility of the *prepare-gpf* and *prepare-dpf* nodes (Figure 3.4). The output of AutoGrid is then forwarded to AutoDock, the final output consisting of a .DLG file for each ligand-protein pair.

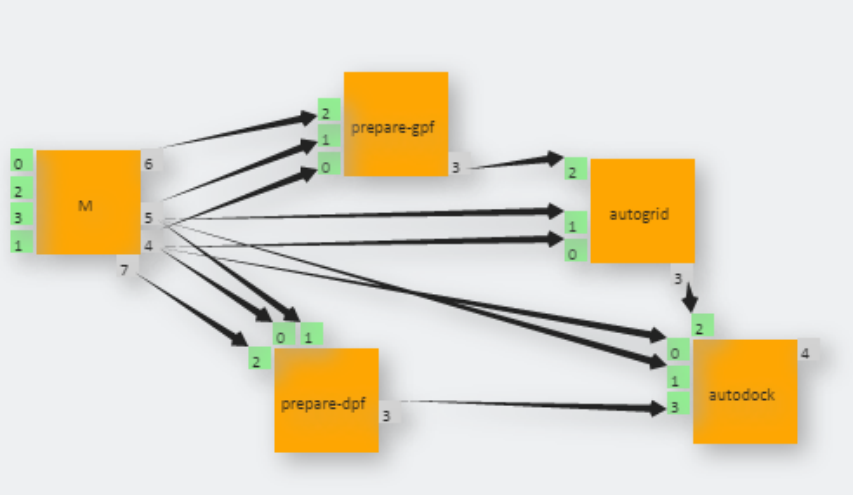

Figure 3.4: WS-PGRADE workflow for the 1st case of the Raccoon extension.

The second option to run AutoGrid "now (and cache the maps)" implies that the action of the nodes prepare-gpf and autogrid would be executed on the same local computer where Raccoon is run. The only nodes that need executing on a DCI would be the prepare-dpf and the autodock node (Figure 3.5). The last option should be used when the scientist has prepared the interaction maps beforehand and can upload them to Raccoon. The same workflow (Figure 3.5) is used with the difference that the input for the AutoDock node would be originally uploaded by the scientists and not created by Raccoon.

The WS-PGRADE workflows used the AutoDock, AutoGrid, prepare_gpf.py, and prepare_dpf.py executables which have been deployed to the cloud using the CloudBroker

Figure 3.5: WS-PGRADE workflow for the 2nd and 3rd case of the Raccoon extension.

deployment process, as described in the user manual [183].

3.5 Results

This extension of Raccoon2 is comparable with similar VS approaches that use cloud computing (more details about these are provided in Chapter 2) and it has some advantages. Unlike wFReDoW, it does not focus on setting up a specific HPC environment, but rather focuses on making cloud computing more accessible for scientists, allowing them to use clouds directly from within a tool that they are used to. Private clouds like Kandinsky (used for AutoDockCloud) aren't available for all scientists, whereas the approach used for this extension of Raccoon2 shows that various types of clouds, including private clouds, can be used. By not developing a new GUI, as it was done in the VENUS-C project, this approach ensures that the learning curve is practically non-existent for a typical scientist who is used to the Raccoon2 GUI. It does this by seamlessly incorporating the execution of the simulations on clouds in the background. Finally, the result analysis capabilities of Raccoon2 can be fully utilised, which was not the case in the aforementioned examples.

To show that the concept can be implemented to run real-life VS simulations on different clouds, biomedically relevant input data was obtained. The receptor was an enzyme called *ribokinase*, which is part of the salvage pathway of nucleotides in the protozoan parasite *Trichomonas vaginalis* (TV). The 3D structure of this receptor has been created by homology modelling. TV causes trichomoniasis, a very common sexually transmitted infection. A set of 130,216 ligands have been obtained from the ZINC database of drug-like small molecules. It is a diverse subset of ligands that may bind and antagonise the receptor. The ZINC database provides a subset of compounds whose members are at least 10% different from any other member. The difference is pre-calculated using "chemical fingerprinting". Such a subset of known small molecules with a molecular weight smaller than 190 represents the 130,216 ligands and is provided by ZINC [185, 186]. The extended

Raccoon2 was tested using these input files, conducting three runs, effectively 130,216 docking simulations in each run.

The UoW OpenStack cloud (based in London, UK) was used to prove that the approach works, and two runs on the commercial CloudSigma cloud (Zürich, Switzerland) were conducted to show the use of different clouds. There are several types of 64-bit (x86_64) instances that can be used in the UoW cloud: small (1-core 2GB RAM), medium (2-core 4GB RAM), large (4-core 8GB RAM), and extra-large (8-core 16GB RAM). Because this experiment was allocated a maximum capacity of 29 instances and 29 processor cores, 29 UoW *small* instances were used. The mean execution time was *26h 35min 52s*. To compare the results of both clouds, 29 instances most similar in type to the UoW small instances were used. The CloudSigma cloud had 32-bit or 64-bit CloudSigma *small* (1-core 1GB RAM) instances, note that they have only 1GB RAM. Two experiments were run using these instance types. The mean execution time was *19h 55min 59s* for the 64-bit and *17h 21min 23s* for the 32-bit instances as shown in Figure 3.6.

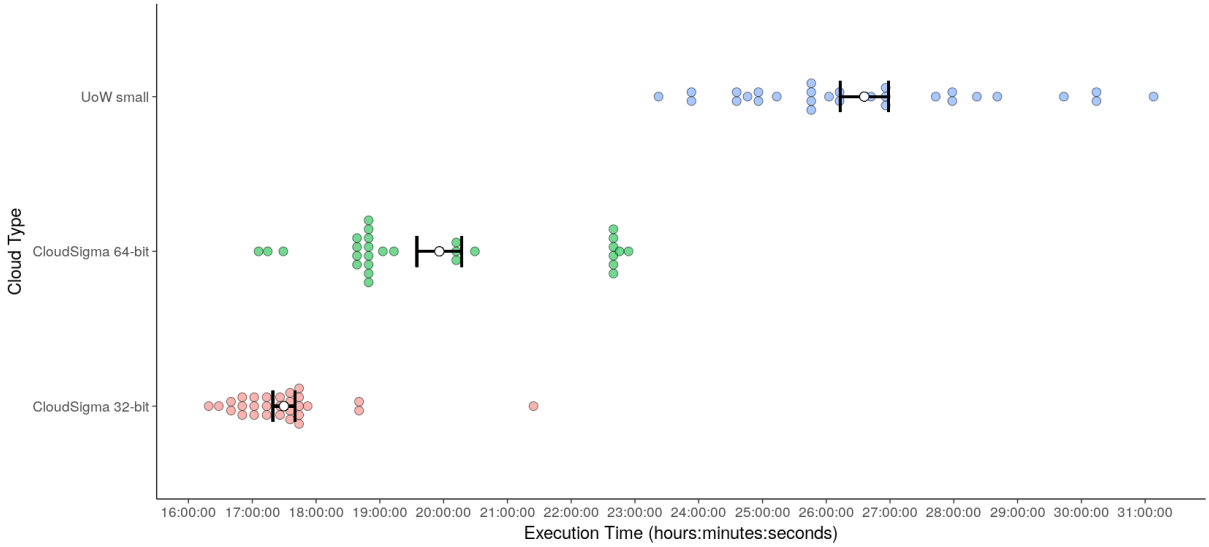

Figure 3.6: Mean, standard error of the mean, and execution times (x-axis) of the 29 jobs on the three clouds (y-axis).

The AutoDock Vina software has been developed for 32-bit machines and as noted on their official website, it is compatible with 64-bit machines [187]. However, it seems that the overhead produced is significant and in general, the recommendation should be to use 32-bit cloud instances for this kind of VS experiments since the average execution time decreased by 12.92%. Furthermore, although the CloudSigma instances had half the memory, due to various performance optimisations in the CloudSigma cloud, they finished the docking significantly faster (on average the 32-bit CloudSigma run was 34.74% faster than the 64-bit UoW run).

At the moment, the UoW cloud can be used by scientists at UoW free of charge. In

Table 3.1: Execution times when increasing instance type and number.

Cloud Instances	Mean Execution Time		Cloud Instances	Mean Execution Time
7 UoW *small*	123h 12min 01s		7 UoW *small*	123h 12min 01s
7 UoW *medium*	75h 35min 16s		14 UoW *small*	61h 31min 01s
7 UoW *large*	51h 47min 29s		28 UoW *small*	31h 29min 14s

general using commercial clouds would incur some costs. As of April 2018, CloudSigma cloud computing prices are $0.0195 per hour for 1-core CPU, $0.007 per GB RAM, $0.1329 per GB SSD storage, and $0.04 per GB of outbound data transfer [188]. Therefore, running our VS on 29 small instances would cost $15.83.

Exploring the potential of using other DCIs As WS-PGRADE/gUSE is connected to other DCIs such as desktop grids, clusters or service grids via the DCI Bridge, the same generic solution and the same workflow mapped to these different resources could be applied to further extend the applicable resources of the experiments. In order to examine alternative DCIs, the experiments were executed on the SZTAKI Desktop Grid (SZDG), a BOINC-based desktop grid [189]. Desktop grids use spare CPU cycles from desktop computers to create a powerful DCI. To show how desktop grids would perform, a WS-PGRADE portal [1] was used to run AutoDock Vina on the SZDG using the same input as above. They were run 5 times, with average execution time: 30h 16min 9s.

Scalability tests In order to show the scalability of this solution, several experiments using the same input files were designed. Firstly, the VS was run using the cloud-enabled Raccoon2, selecting 7 small instances on the UoW cloud. The average time per instance was 123h 12min 1s. Then, the instance type was increased to medium while keeping the number of instances to 7. The average time per instance was 75h 35min 16s. Finally, 7 large instances were used, resulting in average time per instance of 51h 47min 29s (Table 3.1). These results demonstrate reasonable scalability of Raccoon2 when increasing the number of cores inside the instances. The left panel of Figure 3.7 demonstrates the scaling-up when compared to an ideal proportional scaling-up (where doubling the cores should result in half the time).

In a second set of experiments the instance type was kept the same (UoW small) while increasing the number of instances. Namely, 14 small instances were used with the average time per instance of 61h 31min 1s, followed by 28 small instances resulting with average time per instance of 31h 29min 14s. The right panel of Figure 3.7 shows that these results very closely resemble the ideal proportional scaling-up. It shows that although AutoDock Vina has multithreading capabilities, it is faster to run 28 small instances than 7 large.

[1] https://autodock-portal.sztaki.hu/liferay-portal-6.1.0

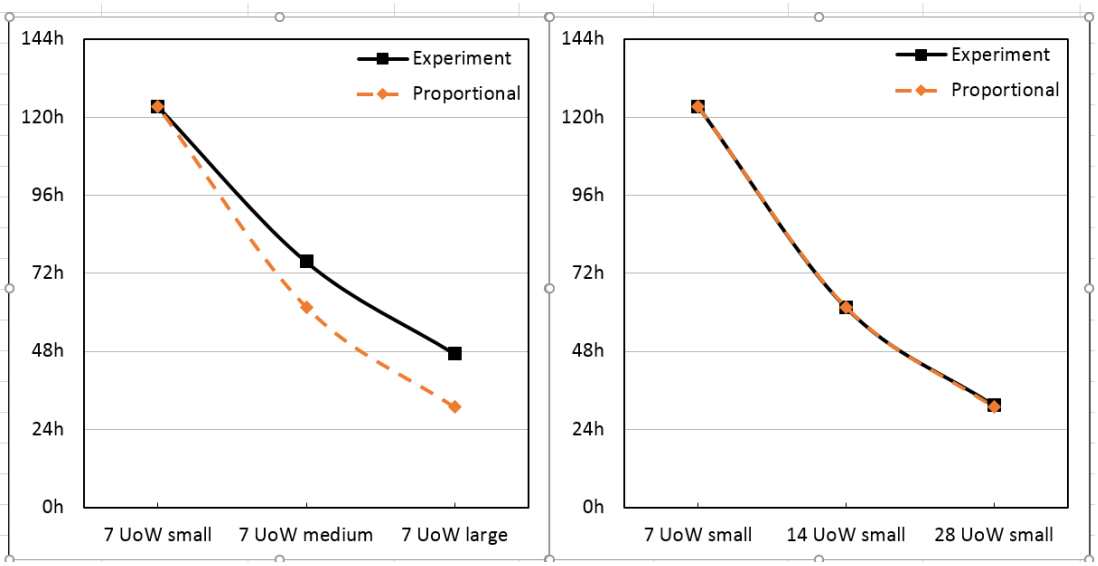

Figure 3.7: Scalability comparison of experimental and proportional cases: increasing the configuration of instances (left), increasing the number of instances (right).

Therefore, to maximise efficiency, it should be recommended to use more, but less powerful, rather than less, but more powerful instances.

3.6 Conclusion

This chapter presented a generic concept to extend domain-specific desktop applications, enabling the execution of simulations on different clouds. A reference implementation of the generic concept has been developed using a desktop application for VS simulations. Several experiments were run to test and evaluate the concept on two different cloud infrastructures and measure the scalability of the solution. Better performance was noticed when using many smaller rather than a few larger instances, and 32-bit rather than 64-bit instances. Although the shown implementation is based on the VS tool Raccoon2, WS-PGRADE/gUSE and CloudBroker, the concept of extending desktop applications to run on clouds is generic.

With this extension, Raccoon2 users can use the same familiar GUI to run their VS experiments on clouds. They no longer require access to a Linux PBS or SGE cluster, which brings down the cost of running large VS simulations, making them more accessible. As shown in the tests, the solution works for different kinds of clouds. At the moment, due to the nature of the gUSE RemoteAPI, result files can only be downloaded to the user's desktop. If instead of downloading the docking results to their own computers, scientists could share the results through a docking result repository, this would enable further analysis, reuse, and collaboration. This is explored in the remainder of the thesis.

Chapter 4

Definition of Conceptual Framework for Systems that Use Molecular Docking Results

4.1 Introduction

In software engineering the term "framework" is often used for a library of pre-made classes and method that can be used when developing software. Chapter 2 showed several examples of open-source libraries that are used in bioinfromatics. However, there is a lack of an abstract conceptual framework that will be independent of the programming language, toolset, or paradigm used. In particular, software systems that use previous docking results can benefit from such a framework. Often software engineers will start the development of a tool from scratch without being aware that they can reuse an existing tool. An abstract description of the element type can be compared to a library of abstract descriptions of existing tools to find a candidate existing tool. Furthermore, a bioinformatician may be uncertain whether an existing tool can be used for a particular scenario. An abstract description of the existing tool in question can be compared to a generic abstract description provided by the framework to show whether the existing tool is suitable for that scenario.

This chapter will summarise the research methodology and the findings that were used to construct a conceptual framework for software systems that use previous docking results. A set of interviews, divided into two parts, formed the primary research activity. The first part included semi-structured interviews with 4 scientists. The resulting requirements were compared to currently available systems for validation. The second part included interviewing one experienced scientist in order to obtain a list of scenarios that would use a repository that stores and manages docking results. Based on these scenarios, a high-level

diagram of the framework was created. A thorough literature review was conducted in order to determine whether existing systems that store docking results (or equivalent simulations) could have been described using the high-level diagram of the framework (Chapter 5). Once this has been determined, a low-level diagram of the framework was created, as well as a textual description of element types and interfaces (Chapter 6). Finally, a formal description of the framework was used to provide generic abstract descriptions of the element types and interfaces (Chapter 6 and Appendix B). Thus, the complete framework is composed of:

1. High-level description using a basic diagram.
2. Low-level, detailed diagram (diagrammatic description).
3. Textual description of element types and interfaces.
4. Formal description of element types and interfaces.

An abstract system can be described using these different views. When determining whether a novel system can be implemented using the framework, or whether an existing tool can fit the framework, the same views can be utilised.

4.2 Research Methodology

This part of the thesis includes a study whose purpose is to define a conceptual framework that can be used when developing software systems that use previous docking results. This should not be limited to the users own previous docking results, but it should include docking results of other users. The framework should not be specific to a single scenario, but generic and useful for a multitude of scenarios. It should describe conceptual elements in a tool-independent way, and enable reusability of elements that are the same between scenarios. If a scenario is suitable for using the framework, a team should be able to follow a specific software development methodology to assist with the use of the framework.

This study contains four segments. The first segment aimed at producing qualitative evidence based on which a high-level framework can be defined. In this part, primary research was conducted through semi-structured and unstructured face-to-face interviews with scientists who have used docking simulations. Through semi-structured interviews with four scientists, the need for a framework was examined. The interviews aimed at testing the hypothesis: “Scientists that use molecular docking require a system that stores and manages molecular docking results.” This was done by analysing the need for a software system that stores and manages previous docking results. The analysis of the semi-structured interviews produced a list of requirements which was compared to currently available systems to

provide reassurance of their validity. On the other hand, through unstructured interviews with another scientist who is an expert in docking, example scenarios that would utilise the framework were examined. These interviews aimed at answering the question: "Which docking scenarios would require a repository that stores docking results?" This resulted in a list of 5 such novel scenarios which require analysis of docking results and cannot be achieved with the current systems. The conceptual similarities of these scenarios were used to design a high-level generic framework.

The second segment aimed at determining whether the framework would have been useful for describing existing systems. The argument being that if the framework can be used for the 5 novel scenarios, and if it could have been used for a large number of existing examples, then it can be used for any system that uses previous docking results. The applicability of the high-level framework description to existing systems in literature was assessed. Secondary research was conducted through a literature review of 14 existing systems, assessing the extent to which the high-level framework can be used to describe them. The fact that the 5 scenarios identified as a result of the first part as well as the 14 examples from literature can be described using the high-level framework, shows that the framework can be used in general and is not tailored for a specific scenario.

In the third segment, the framework was extended with low-level diagrammatic, textual, and formal descriptions. The aim of the third segment was to enable a more formal assessment of the applicability of the framework to a new scenario about to be implemented. Given a new scenario, a formal description will allow future developers to prove whether the framework can be used or not. This can be done by describing the new scenario formally and comparing it to the formal description of the framework. An existing tool can be described formally and compared to the generic description of an element type to determine whether it can be used. Furthermore, the formal description of an element type can be compared to existing tools to determine whether there is a need for creating a custom-made tool. The third segment included a definition of a software development methodology. The methodology, described in detail in Chapter 7 shows the type of team that is required to implement a scenario and the actions that should be undertaken when developing a system that uses the framework.

Finally, the fourth segment included an experimental evaluation of the framework and methodology. A total of 19 systems were described using the high-level view of the framework, 14 of those were identified in the literature, while 5 in the interview process. Of the 5 scenarios identified in the interview process, 3 were implemented using the methodology and the low-level views of the framework (Chapter 8). These scenarios were selected because they show that the framework allows easier implementation through reuse of elements and easy integration of new elements.

4.3 Main Findings of Primary Research

The interviewees for the first segment of this study were five scientists from different backgrounds, with various degrees of experience with docking simulations. The first set of four scientists represented early-career researchers, while the fifth scientist was a very experienced academic and expert in the area of docking. All scientists were completely or partially based in London where all the interviews took place. Using a small population in interview-based requirements analysis is not uncommon in the area of docking. For instance, the AMC Docking Gateway based the requirements gathering solely on one interviewee, an expert in the field [116]. However, it is worth noting that the aim was not to interview a representative sample of the global population of scientists that use docking. The aim was to collect enough data to be able to design a useful framework and methodology for developing software systems that require a docking result repository.

4.3.1 Need for a system to store and manage docking results

The first segment of this study included analysis of the interviews. Participants were asked to consider voice recording (3 out of 5 agreed; notes were taken for the remaining 2 participants). The transcribed voice recordings and notes were analysed using NVivo v11 [190]. A summary of the analysis of the notes and the transcribed recordings is provided in Appendix A. The participants of the semi-structured interviews are pseudonymously referred to as scientists A, B, C, and D. The participant of the unstructured interviews is referred to as scientist E.

Table 4.1 summarises the data analysis process of the semi-structured interviews with the first four scientists. The semi-structured interview questions asked the participants to comment on their experience with: running docking, using data storage and management techniques and/or existing systems. All interviewees ranked functionalities of a system that would store docking results and their provenance. Scientists A, C, and D used a scale from 1 to 10 (note that numbers in parenthesis are implicit interpretations of the interviewees' statements), while Scientist B preferred to described the functionalities with words only. This resulted in a list of requirements for a software system that stores and manages docking results. A (non-exhaustive) list of currently available systems was matched to the list of requirements obtained from the interviews in order to validate them.

The conclusion of the semi-structured interviews is that scientists have not used a system specifically designed to store and manage docking results. However, there is a clear need for such a system, as all interviewees found several useful functionalities. The functionalities that were found to be useful can be summarised as requirements.

Table 4.1: Summary of interviews with interviewees A-D.

Scientist	Docking Experience	Storing and Managing Data	Using Provenance
A	100,000s docking simulations using AutoDock Vina, one docking takes 30 seconds to 30 minutes each. Used their own computer or a GPU cluster. 40 MD simulations using AMBER [121] to post-process the docking results taking into account the flexibility of the receptor [191].	Obtained ligands from public databases e.g. PubChem [17] and ZINC [16], and created homology models of the receptors, sometimes followed by short MD. Always refines and fixes the input file. Stores all input files on own computers - has encountered problems as the amount of files gets overwhelming and one cannot locate past files easily. One should store the originally downloaded file, as well as the refined input file. Stores the configuration and intermediate files in the same location. Result files, particularly for MD, are very big which is problematic.	Hasn't used provenance management systems. Providing a publication along with the docking results would be very important to view what the particular simulation was used for and why. Ranked other functionalities of such a system as (scale 1-10, 10 being most useful): • Automatically redoing docking: 7. • Automatically redoing docking on the cloud: good and important to have (7). • Comparing results from past simulations: 8-9. • Contacting the people that performed the simulations: 3-4. • Downloading the input, intermediate and result files: 9, 8-9, 9-10 respectively. • Obtaining more details about the software tool used: 7-8. • Viewing very old simulations: 2.

Continued on next page

Table 4.1 – continued from previous page

Scientist	Docking Experience	Storing and Managing Data	Using Provenance
B	More than 2 years ago, on a high performance cluster at their institution: MD using Gromacs [104], CHARMM [192] and NAMD [102] and docking using AutoDock and docking software developed at their institution. Docking and MD simulations should always be done in conjunction. Typical MD would run between 2-24 hours, while docking 4-12 hours, depending on the configuration.	Created models of the input structures, received models from colleagues, or got them from the on-line database: PDB [14]. Tools to refine the models or to fix some issues included Swiss PDB Viewer [193], or their own scripts. Always stores the original "raw" files, the refined version in a separate folder, a log file indicating the time and date of each step from start to completion. Including parameter or configuration files (however these are not as important and not backed up). Several copies of the input files stored on their own computer, and a cloud storage system. All other files stored on the cluster.	No academically-friendly provenance management system that fits to their job types. Additionally, simulation results of others are trusted, but if they have not translated into good results on a paper, then one may question them and need to redo the simulation. Therefore, storing information regarding a paper is important. Adding notes regarding this would be very useful. A useful addition would be to store well-defined log files which would contain enough information to see which step was used with which files. Commented on importance of functionalities of an ideal system: • Automatically redoing simulations to verify the results: absolutely important. • Storing files (by order of importance): input, output, intermediate. • Storing information about people that performed the simulations (name, date and time): absolutely important. • Contacting the creators: only in case of collaborations.

Continued on next page

Table 4.1 – continued from previous page

Scientist	Docking Experience	Storing and Managing Data	Using Provenance
C	About 200 molecular docking simulations in the past year, using AutoDock 4.2 on their own computer. Depending on the size of the ligand, it would take from 20 minutes to one hour per docking - they have been running dozens of simulations at one time, leaving the computer to run overnight.	One fifth of the input receptors had solved crystal structures and came from the PDB, while the majority were homology models created by using SWISS-MODEL [194]. The SMILES codes of the ligands from ChemSpider [195] were used to create their 3D structure. Apart from visually checking them, no major changes were done to the input files, only the PDB ID and the refined version of the input file was stored - not the original. Storing the intermediate grid maps was important so one can go back and redo the simulation with a different ligand or other minor changes. The result files were also stored, filtered and analysed to produce additional files with some conclusions.	Has not used a provenance management system, concerned about intellectual property - would not use if others could publish a paper based on their results. Docking results should be published only once the author publishes a paper. If the docking was done on the system, it should not claim ownership and prevent the scientist from publishing a paper using them. Ranked other functionalities (scale 1-10, 10 being most useful): • Automatically redoing docking to verify the results: 7 - more useful if done on the website, without downloading or installing anything. • Comparing past simulations: 7. • Contacting the creators of results: 6. • Downloading the input, intermediate and result files: input (9) more important than results (7). • Docking tool details: very useful (8-9). Additionally, searching for all ligands docked to a particular protein, or vice versa, and their docking results. Searching a ligand-receptor pair should return a list of species, and if different tools have been used, all the result formats.

Continued on next page

Table 4.1 – continued from previous page

Scientist	Docking Experience	Storing and Managing Data	Using Provenance
D	Used molecular docking extensively in a recent project involving around 1100 ligand and 4 targets, running them once with AutoDock Vina and once with DOCK v6 [81], for a total of about 9000 simulations. Each individual simulation took 5-10 minutes to complete.	Computed their own 3D models of the structures for the ligand files, starting from a drawing of the chemical formula drawn in ChemDraw [196] or MarvinSketch [197], then using Avogadro [198], energy minimisation and structure optimisation. The PDB was also used to get structures for the targets, they were processed before the docking, differently for AutoDock Vina and for DOCK. ZINC was used for structures of the ligands. The original and the altered input files, as well as the intermediate files were stored on their own computer and a remote storage in their affiliated institution. Output files were filtered, using the built-in filtering function of Chimera [199]. The results were also manually checked for inconsistencies using Chimera and ViewDock [200].	Have not used, nor searched for an available provenance management system. Feels that administrators of such systems should pre-verify the results. Ranked functionalities of such a system as (scale 1-10, 10 being most useful): • Automatically redoing docking on one's own computer: useful, but rather difficult as there are many tools that need to be installed and configured (6). • Docking on cloud: if everything is prepared as a virtual machine, it would be much easier to redo them (7). • Comparing past simulations: 9. • Contacting the people that performed the simulations: 10.

List of requirements and validation As a result of the four semi-structured interviews, the following list of requirements was compiled. It is based on what scientists would use the docking results storage and management system for, and what they would store.

1. Search all docking results based on a receptor, a ligand, or both.
2. Re-do simulation on the cloud.
3. Re-do simulation on a local computer.
4. Explore details about the creator, date and time of the simulation.
5. Compare your simulation results with past simulations (your own or someone else's) using the same or different software.
6. Store and download intermediate files (ones between steps), log files (execution summaries), result files, and structure of molecules (input files).
7. Store a link to a peer-reviewed paper published based on the docking results.

More details about the reasoning behind compiling this list is shown in Appendix A. As shown in Figure 4.1, this list was used to analyse several existing software systems and assess to what extent these requirements are fulfilled (based on literature evidence and personal experience). The aim of this activity was to see how reasonable the requirements are and if scientists can use an existing system for any of them. More information on the chosen currently available systems can be found in Section 2.8.

As a result of this analysis, it can be deduced that the capabilities of the MoSGrid system fulfil most of the requirements laid out by the scientists. In the case where it doesn't (linking a peer-reviewed paper to each docking), extending MoSGrid seems possible. MoSGrid allows scientists to redo simulations on the German grid (D-Grid), but it could be connected to clouds similarly to the CloudSME portal, since it uses WS-PGRADE/gUSE. In its current implementation, MoSGrid allows scientists to download their own workflows after which they can be rerun on one's own computer if a WS-PGRADE system is connected, or the individual jobs can be rerun manually. Adding provenance information to MSML has already been identified by the MoSGrid authors as a potential improvement [143, p. 1755].

The four semi-structured interviews showed that there is a need for a system that stores and manages docking results. In general, the scientists were not acquainted with the term "provenance", but it would be beneficial if this system managed the provenance of docking results as well. The scientists point out 10 requirements (if the 6^{th} requirement for storage from the list above is split). Most requirements are fulfilled to some degree by currently available systems. For instance, MoSGrid completely fulfils 7, partially fulfils 2, and does

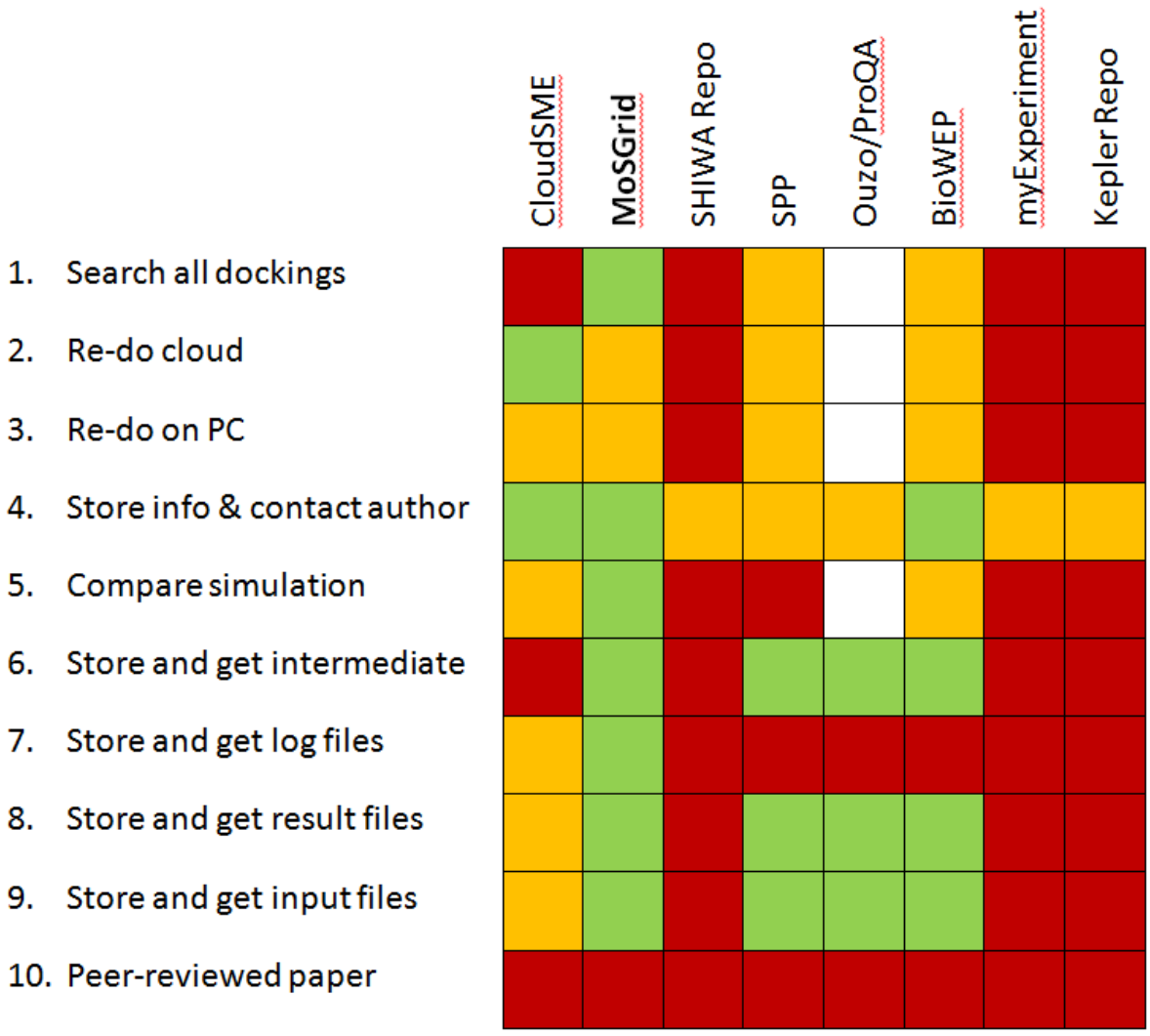

Figure 4.1: Summary of the fulfilment of requirements from the interviews: *green* signifies fulfilled, *amber* partially fulfilled, *red* not fulfilled at all, and *white* lack of information.

not fulfil 1 requirement. This shows that the requirements are reasonable and have been echoed by scientists internationally.

4.3.2 Novel scientific scenarios using docking results

Once it became clear that there is a need for systems that store and manage docking results, an obvious step was to create one. However, instead of creating a software system from scratch, the possibility of defining a generic framework was explored. If there are several scenarios that require a docking results repository, and they cannot be fulfilled with the currently available systems, a framework can be derived from the conceptual similarities between these scenarios. This framework can then be used to implement these and any similar scenarios. In the second segment of this study, interviews with a fifth interviewee

(Scientist E) were conducted, aiming at providing a list of novel scenarios which require a repository that stores and manages docking results. These scenarios represent software systems that assist in making a decision based on analysing previous docking results which have been stored in a repository. Comments made by Scientists A, C, and D in the semi-structured interviews are useful for 3 of the 5 scenarios that were identified. The five scenarios, which will be used to define the framework, are listed below.

1. Suggest a ligand-protein pair that should be used in the next molecular docking, based on protein similarity and previous results.

Once a docking simulation has been conducted, scientists analyse the results. If the results for the particular protein-ligand pair are not interesting, the scientist would search for a similar protein and attempt a new set of docking simulations. The similarity between protein structures can be used to find a similar protein (as noted by Scientist E).

2. Filter results suitable for laboratory experiments, based on ligand properties.

When scientists conduct large-scale docking simulations, the results need to be filtered during the analysis. Usually, a scientist searches for an interesting protein-ligand pair to examine in a wet-lab experiment. The interesting protein-ligand pair may include a ligand which seems useful, but in fact is not usable in the laboratory. This may be because it is not purchasable due to its toxicity, or various other properties (noted by Scientist E and mentioned by Scientist A).

3. Find off-target drugs, based on deducing if the binding is on an active site.

Off-target drugs are drugs that were designed to bind to a receptor, but additionally bind to another receptor of the same or a different organism and produce often unplanned effects. The steps necessary to find off-target drugs are: explore if the drug binds to the active site of one or potentially a large range of proteins that are not the primary target, search for similar drugs that may have the same effect, and conduct wet-lab experiments. A subset of all the human proteins with a solved structure would be needed if humans are of interest (as noted by Scientist E and mentioned by Scientist C). From the computing point of view, the first step is the most relevant. It requires a method to estimate if a docking between a ligand and protein is on an active site. Analysing past docking results can be used to provide this.

4. Enable verification of the docking methodology and learning from previous docking.

Storing molecular docking results and their provenance, even if the results are negative or "null results" is important, in these two cases (as noted by Scientist E):

(a) Another scientist may run the same molecular docking simulation and expect to get useful results. If they don't, they may suspect that they are conducting the docking wrongly. Comparing their input and output files with ones from another scientist

will enable them to verify their docking methodology and realise that it is expected to get results that are not useful.

(b) Scientists with little or no experience in running docking simulations can learn quicker if they view previous docking input and output files, regardless of the usefulness of the final results.

5. Compare results from different molecular docking tools. Comparing the docking results of the same input files, but using a different docking tool is important. This will enable scientists to determine if there is a significant difference in the results when using different tools, which may be relevant for further wet-lab experiments (as noted by Scientist E and mentioned by Scientists C and D).

4.3.3 High-Level description of conceptual framework for systems that use previous docking results

All five scenarios begin with the scientist using an environment to conduct molecular docking or VS simulations. A repository specifically designed to store and manage docking results is needed by all scenarios too. Depending on the scenario, the stored previous docking results are then processed by one or more elements. Some scenarios include the same type of processing, for instance, Scenario 1 and 2 need a tool to assess whether a docking result is good or not. Scenarios 2 and 3 require reading data from an element that is external to the system. All scenarios include an element that groups and summarises the data to make conclusion or decision. Based on this analysis, similar elements can be grouped or generalised into five element types.

• Molecular Docking Environment (MDE): The MDE includes the software tool used to run the docking itself, and may include additional components to connect to a DCI. It could be as simple as running a command on one's local computer, or more complex such as running a scientific workflow on a DCI.

• Molecular Docking Results Repository (MDRR): The docking results from the MDE should be passed to a repository for storage and management. A user could have access not only to one's own, but also to previous results created by other users. This results repository could store information about the final decision made by the entire system.

• Additional Tool (AT): This is a generic element type that describes a tool which takes one or more docking results from the MDRR as input and produces a calculation. The AT can refer back to other docking results stored in the MDRR, communicate with another AT, or refer to data stored externally.

• Additional Data Source (ADS): This element type describes a tool such as a database that contains relevant data for the final decision. This could be an external database that does not store docking results, but other types of data.

• Decision Maker (DM): All the information processed from the various ATs are passed to an element of the type DM. A DM groups and analyses the calculations done by the ATs, and then makes a decision based on these calculations. It may use previous docking results stored within the MDRR in the decision making process.

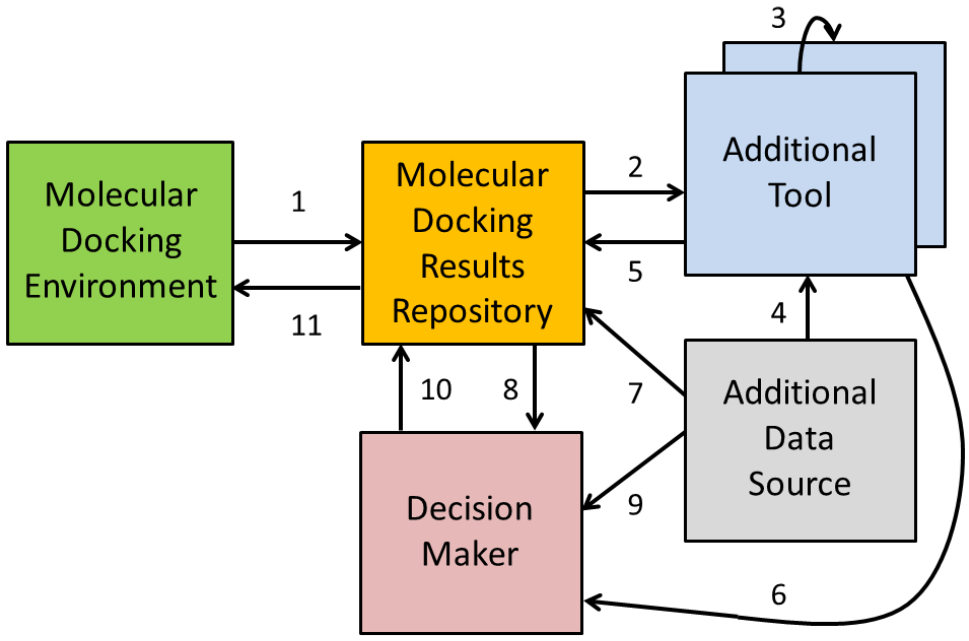

Figure 4.2: Basic diagram of the framework.

Based on the description of these element types which was derived from the scenarios, a high-level diagram of the framework can be proposed (Figure 4.2). The numbers signify the order and flow of events through elements of different element types.

1. A scientist uses an MDE to conduct the docking and upload the result to the MDRR.
2. The MDRR sends the results to one or more ATs.
3. An AT may communicate with one or more other ATs.
4. An AT may look up data stored in an ADS.
5. An AT may require additional previous docking results as input for its calculation.
6. An AT would provide its calculation results to the DM.
7. The MDRR may use data from an ADS directly.

8. Previous results from the MDRR may be used by the DM.
9. The DM may use data from an ADS directly.
10. Once the analysis is complete and the decision made, it can be passed to the MDRR.
11. The decision is passed to the MDE to visualise it.

Based on this flow of events, 11 interfaces between pairs of elements can be described. When the communication between the user and the MDE is taken in consideration, this framework contains a total of 13 interfaces. A more detailed explanation about the interfaces follows in Section 6.

4.3.4 Verification of high-level view with novel scenarios

The high-level view of the framework contains a high-level or basic diagram. Effectively, the hypothesis is that this framework, starting with the basic diagram, is suitable for all systems that require analysis of previous docking results. To show that this diagram can be used for the 5 scenarios identified as part of the primary research, each of the scenarios is described using the basic diagram of the framework. This shows that each specific scenario can be derived from the generic framework. For each scenario a title, scientific goal, description, flow of events, and a diagram is shown.

Scenario 1 Suggest a ligand-protein pair that should be used in the next molecular docking, based on protein similarity and previous results.

Scientific Goal: Identify the next docking of interest, based on a ligand already docked with a similar receptor.

Description: In this scenario a software system would analyse previous docking results and look for similar proteins to the one currently used. Based on the past docking results of these similar proteins, the system will suggest a new protein-ligand pair that would be an interesting candidate for docking. A protein similarity tool should search for similar proteins within the repository of results of previous docking simulations. This scenario can suggest docking the current protein with a ligand which has been docked successfully to the similar protein. In order to do this, an analysis of the past docking is required together with a method to define a docking result as "successful". Any existing tool capable of running VS simulations can be an **MDE**. For instance using the extended version of Raccoon2 developed as part of this thesis has several benefits as described in Section 3. A tool that can store and manage docking results can be an **MDRR**. A protein structural alignment tool needs to be an **AT** in Scenario 1. The tool DeepAlign could be a good choice, as

explained in Section 2.3.1. Another AT should assess whether the structural alignment score is sufficient to declare two proteins as similar. A simple solution would be to create a custom-made tool to compare the value of DeepAlign with a threshold input from the user. A third AT is needed to assess how good a docking is. Similarly, a custom-made tool can do this based on a user input threshold. Finally, a **DM** needs to summarise the results of the ATs. The DM needs to be a tool specific to the scenario. So in Scenario 1, and in general, a specific custom-made DM is required.

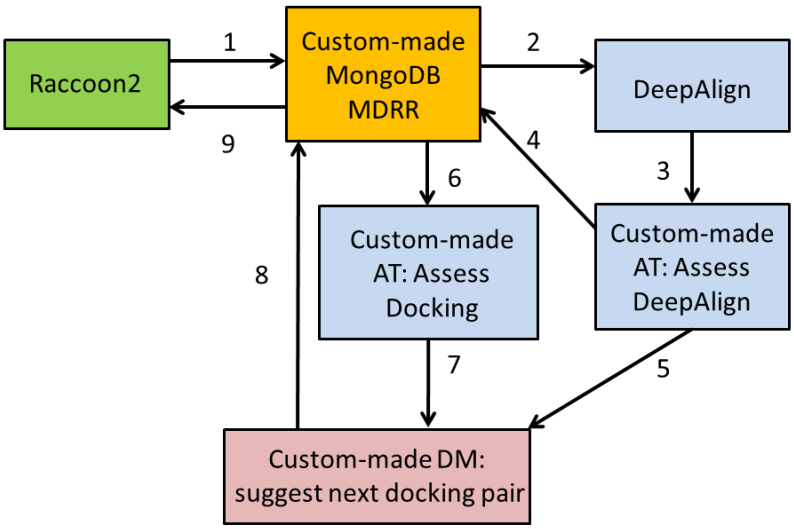

Figure 4.3: Basic diagram of Scenario 1.

Flow of events

1. Raccoon2 executes the molecular docking and the results are uploaded to the MDRR.
2. The MDRR sends the receptor pairs to the DeepAlign AT.
3. The results of DeepAlign are assessed by the custom-made AT.
4. It sends the results to the MDRR.
5. It also sends the results to the DM.
6. All past docking results of similar receptors are sent to be assessed.
7. The "good" docking results are sent to the DM.
8. The DM combines the results from the ATs, and suggests which protein-ligand pair to dock as a next step - the suggestion is returned and stored in the MDRR.
9. Finally, the suggestion is presented to the user.

Scenario 2 Filter suitable results for laboratory experiments, based on ligand properties.

Scientific Goal: Improve estimation of ligands' viability before conducting wet-lab experiments.

Description: VS simulations contain docking results that can be interpreted as a list of ligands sorted by how likely they are to bind to a protein. A database with relevant information about every ligand will be beneficial for further filtering the results to get ligands that are more viable for the wet-lab experiment. One option is to select the "good" docking results, and consult an external database of molecular properties. This database would store molecular properties and information about ligands that cannot be easily calculated from the ligands structure. As in Scenario 1, any existing tool capable of running VS simulations, e.g. the extended version of Raccoon2, can be an **MDE**, any tool that can store and manage docking results, can be an **MDRR**. Any tool that can assess how good a docking is can be an **AT**. For instance a custom-made tool like the one in Scenario 1 can be created. Any external database which stores molecular properties, such as the Components database of PubChem [17], can be an **ADS**. Finally, a custom-made **DM** to summarise the results of the particular AT and ADS can be created.

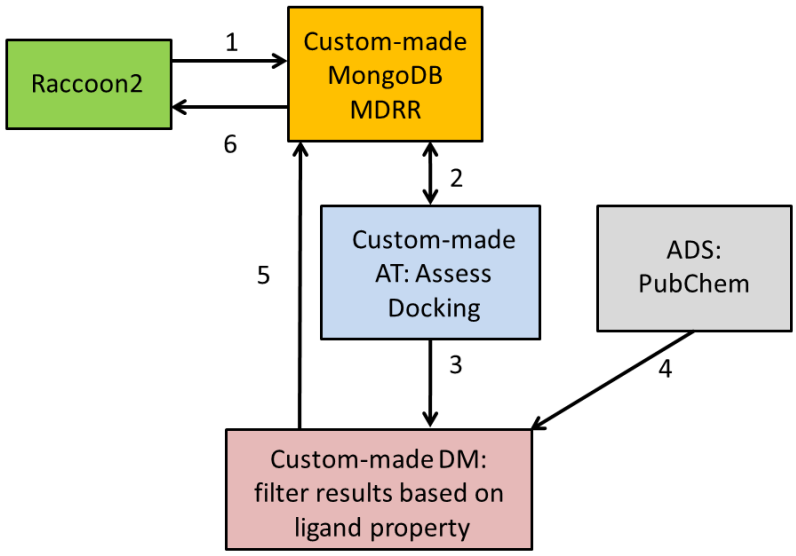

Figure 4.4: Basic diagram of Scenario 2.

Flow of events

1. Raccoon2 is used to execute the docking and then upload the results to the MDRR.
2. The results are filtered into "good" by a custom-made AT.

3. The "good" results are passed to the DM.
4. The DM also consults an ADS (PubChem) to obtain specific ligand properties.
5. Results are grouped and analysed before the DM sends the decision to the MDRR.
6. Finally, the filtered results are displayed in the Raccoon2 GUI.

Scenario 3 Find off-target drugs, based on deducing if the binding is on an active site.

Scientific Goal: Use molecular docking to find off-target drugs.

Description: An example of an off-target drug is a heart drug for humans which has been docked to the active site of a protein of a protozoan (a single-celled organism). By binding to the active site of this protein the drug inhibits the protein, meaning it stops it from performing its function. Looking for other drugs that have a similar accidental (so called off-target) effect on human proteins is very interesting from a biomedical point of view. In order to do this, a software system will need to use a tool to analyse a binding site between a ligand and a protein and determine if the binding site is the protein's active site, therefore deducing if the ligand would be an inhibitor. The active site can be identified based on previous results, in a "crowd-sourcing" manner, assessing where previous ligands have docked to a particular protein. As in Scenario 1 and 2, any existing tool capable of running VS simulations, e.g. the extended version of Raccoon2, can be an **MDE**, and any tool that can store and manage docking results, can be an **MDRR**. An **AT** to identify active sites based on previous results is needed. To the best of the candidate's knowledge, such a tool does not exist, so a custom-made AT would be required. Any database that contains the structures of proteins with bound ligands, such as the wwPDB [14], can be used as an **ADS**. Finally, a custom-made **DM** to summarise the results of the specific AT and ADS can be created.

Flow of events

1. The docking results are uploaded to the MDRR.
2. A list of protein-ligand pairs is passed to the proposed active site identification tool.
3. This tool searches the MDRR, looking for other ligands that have been docked to this protein on that binding site. If a certain number of ligand have been bound to a binding site, this tool can conclude that it is an active site. This is a "crowd-sourcing" way to see if a binding site is an active site. It then passes this information to the DM.
4. The DM also receives relevant information from the wwPDB to assist in deciding if a binding site is an active site.

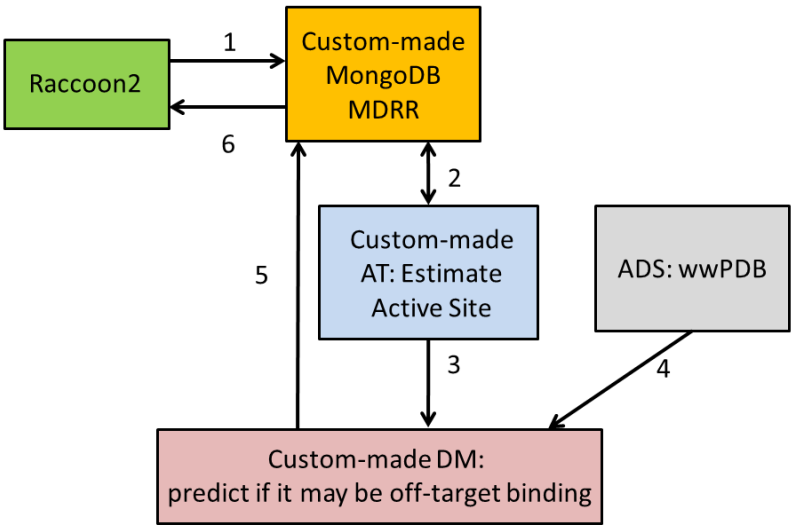

Figure 4.5: Basic diagram of Scenario 3.

5. The DM makes a decision based on all the information and sends it to the MDRR.
6. Finally, the information should be visualised in the Raccoon2 GUI.

Scenario 4 Verify your docking methodology and learn how to conduct docking.

Scientific Goal: Verify that the docking methodology is correct, and/or learn how to conduct docking correctly from previous docking experiments.

Description: Scientists may get unexpected docking results and may not be able to determine whether this is because of a mistake in their docking procedure. Consulting a repository of previous docking results can help determine this. Scientists should be able to upload all their input and output files and search the repository for previous results with the similar input files. This scenario also enables scientists who are not very experienced to learn how to conduct docking correctly by observing previous docking experiments. In this latter case, the same approach would be used, where scientists upload their input files and search through the repository of previous results. As in Scenario 1, 2 and 3, any existing tool capable of running VS simulations, e.g. the extended version of Raccoon2, can be an **MDE**, and any tool that can store and manage docking results, can be an **MDRR**. Scenario 4 requires five **ATs**. As in Scenario 1, a protein structural alignment tool, such as DeepAlign, is needed. Another AT should assess whether the structural alignment score is sufficient to declare two proteins as similar. This could be a custom-made tool that uses a threshold input by the user. The third AT should be a tool that is capable of comparing ligands. Any tool that fits this description can be used, for instance Section 2.3.2 shows

that LIGSIFT is a good tool to find structural similarity between ligands. A fourth AT needs to assess the ligand similarity results, this could be a custom-made tool that uses a threshold input by the user. The fifth AT should be a tool that compares docking configuration files. This tool would be specific to the docking algorithm, and in the lack of an existing solution, a custom-made tool could use a user-provided threshold to assess whether the configuration of two docking experiments is similar. Finally, a custom-made **DM** to summarise the results of the ATs can be created.

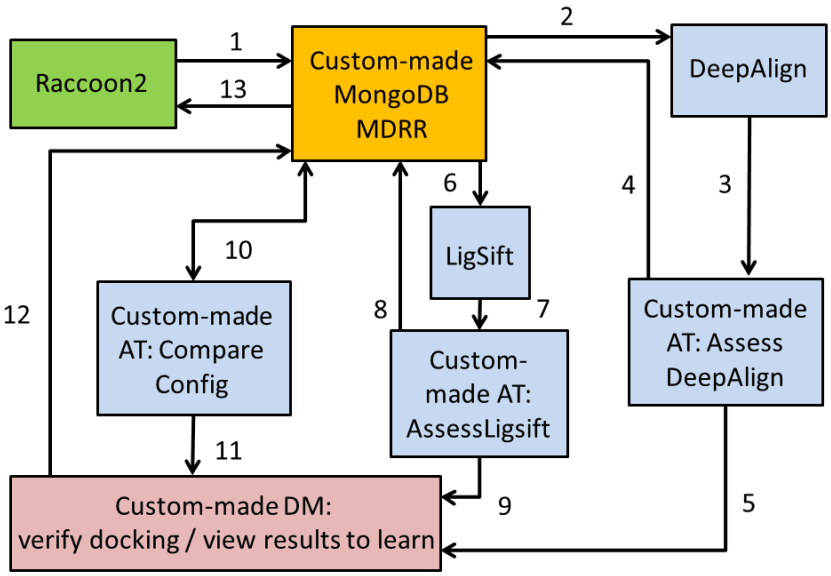

Figure 4.6: Basic diagram of Scenario 4.

Flow of events

1. Raccoon2 is used to run a VS and upload docking input and output to the MDRR.
2. Similarly to Scenario 1, the MDRR sends the receptor pairs to the DeepAlign AT.
3. The results of DeepAlign are assessed by the custom-made AT.
4. This AT sends the results to the MDRR.
5. It also sends the results to the DM.
6. The MDRR sends ligand pairs to the LIGSIFT AT.
7. The results of LIGSIFT are assessed by the custom-made AT.
8. This AT sends the results to the MDRR.
9. It also sends the results to the DM.
10. The MDRR sends config files used by similar receptors and ligands, to a custom-made config file comparison tool.

11. The config file comparison AT sends the results to the DM.
12. The DM makes a decision based on all the information and sends it to the MDRR.
13. The decision is reported back to the Raccoon2 GUI.

Scenario 5 Compare docking results of different docking tools.

Scientific Goal: Compare results of a docking between the same ligand and receptor, but with a different tool.

Description: Scientists often need to compare results from different docking tools. If there is a significant difference in the results of two docking tools then one of the tools may not be accurate for that example. Comparing results of different tools will help scientist decide which tool to use. As in all previous scenarios, any existing tool capable of running VS simulations, e.g. the extended version of Raccoon2, can be an **MDE**. However, Raccoon2 provides docking with AutoDock Vina only, so this scenario would require at least one more MDE in order to obtain a repository with docking results of more than one docking tool. For instance, the web application DOCK Blaster [202] which uses the docking tool DOCK can be used as the second MDE. The choice of these tools is irrelevant for the framework. As in the previous scenarios, any tool that can store and manage docking results, can be an **MDRR**. The **AT** in Scenario 5 could be any tool that understands the output format of multiple docking tools (in this example AutoDock Vina and DOCK). This tool would provide a result of the comparison of two different output formats. Depending on the docking tools in question, a custom-made AT may need to be developed. Finally, just as in the previous scenarios, a custom-made **DM** is needed.

Flow of events

1. Raccoon2 and DOCK Blaster are used to run a VS and upload results to the MDRR.
2. The MDRR selects past results with the same input files, but with a different docking tool and sends them to the custom-made docking-result comparison AT.
3. The results from this comparison are passed to the DM.
4. After summarising the comparison results, the DM sends the decision to the MDRR.
5. The decision is reported to the GUI of Raccoon2 (if the same user runs the scenario there would be no need to send it to DOCK Blaster).

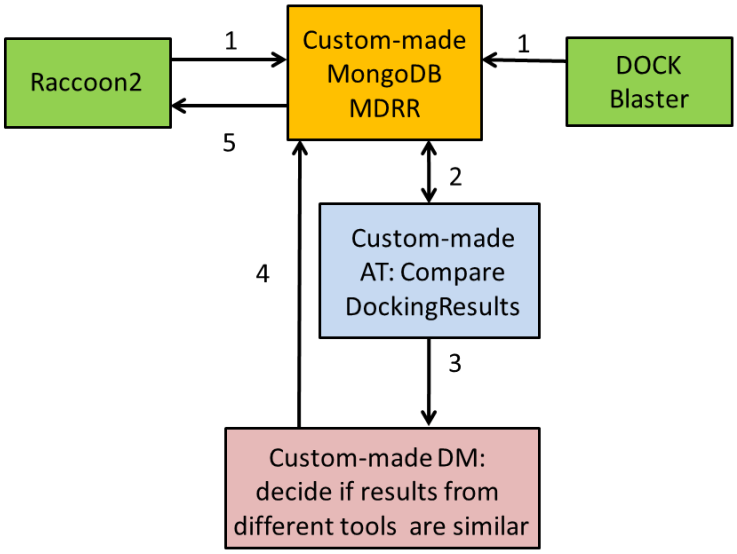

Figure 4.7: Basic diagram of Scenario 5.

4.4 Conclusion

This chapter outlined the research methodology and the main findings of the primary research. The primary research included interviews with five biomedical scientists which were used as a basis for gathering requirements for a system that would store docking results and associated metadata. Five novel scenarios emerged as a result of the interviews and were described in this chapter. The similarities among these scenarios were used to define the high-level description of a conceptual framework which consists of five elements (MDE, MDRR, AT, ADS, and DM) and the interfaces between them. The remaining segments of the research methodology will be covered in the following two chapters.

Chapter 5

Main Findings of Secondary Research

5.1 Introduction

The first segment of the study, as described in the previous chapter, showed that the framework can be used for the 5 novel scenarios. The second segment aims at determining whether the framework would have been useful for describing existing systems. If it could have been used for a large number of existing examples, then one can assume that the framework can be used in general, for any system (that is, any system that uses previous docking results). This chapter shows a literature review of 14 existing systems and a reflection of how the basic diagram of the framework could have been used to describe them. Existing environments for VS can be divided into VS pipelines (6 examples shown here), and workflow-based docking systems (4 examples). An additional group of existing systems that do not use VS or docking, referred to as docking-equivalent systems, can also be described with the basic diagram of the framework (4 examples).

5.2 Verification of high-level view with existing systems

VS pipelines are systems that contain a set of scripts or tools to be used in a particular order. These pipelines can be set up in order to explore the interaction of a particular molecule and are usually not used through science gateways. VS pipelines consist of: methods to prepare the input files for docking, a docking algorithm, a procedure to store the docking results, and methods to further process the docking results. Workflow-based docking systems use scientific workflow management systems to define workflows which: prepare the docking input files, conduct the docking using a docking algorithm, and analyse the docking results or provide further calculations. They include storage of

the docking results either through the workflow management system, or by providing a custom-made storage layer. The workflow-based docking systems are usually accessed via science gateways. The element types defined in the basic diagram of the framework are specific for systems that use docking. Docking-equivalent systems use a different type of bioinformatics tools and may or may not use workflows. This section will show that they provide elements that can be viewed as equivalent to the framework's element types.

5.2.1 Virtual screening pipelines

A total of 6 VS pipelines are presented in this section. A short title, description, elements that belong to the framework's element types, and a diagram are shown for each example.

Title: Docking and MM/GBSA rescoring pipeline.

Description: Zhang, Wong, and Lightstone (2014) [203] present a drug discovery pipeline composed of docking and Molecular Mechanics / Generalised Born Surface Area (MM/GBSA [204]) rescoring. Their system includes a pre-docking preparation step, docking with a tool called VinaLC [205] on an HPC cluster, followed by rescoring of the top 20 poses for each ligand-receptor complex. They describe the specific parameters and programs used in the steps that prepare the receptor and the ligand for docking, and the parameters for the docking itself. These two sets of steps form an MDE. The user can upload receptor and ligand files, the files are prepared, and docking is conducted. The top 20 poses of each docked ligand are stored after a particular post-docking step adds non-polar hydrogen atoms to them. These ligands are stored in one PDB file together with the corresponding receptor. This storage represents a type of MDRR with relatively limited capabilities based on storing files on a file system. The energies between the ligand and receptor are rescored using the MM/GBSA method and the top 20 docking poses are reranked. The rescoring tool is one AT and the reranking represents another AT (Figure 5.1).

MDE: Set of tools to prepare the receptor and ligand, and run docking with VinaLC.

MDRR: Post-processing and storing of top 20 poses of each docked ligand.

ATs: Rescoring and reranking.

Title: Docking, MD and MM/GBSA pipeline to analyse Nelfinavir.

Description: Xie, et al. (2011) [206] have developed a pipeline to find the off-target activity of Nelfinavir using docking, MD and MM/GBSA calculations. This pipeline begins with a pre-docking step using the binding site comparison tool called SMAP [207]. The

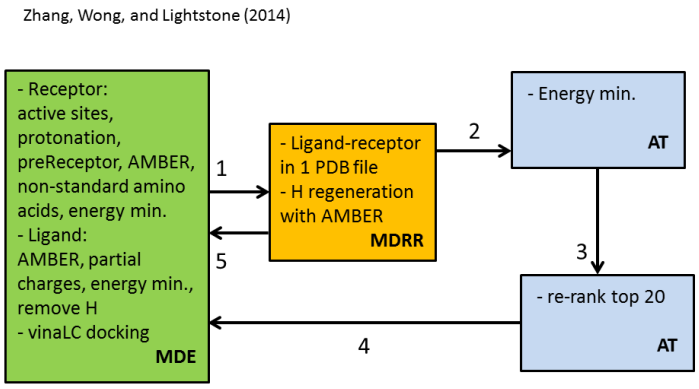

Figure 5.1: Basic diagram of Zhang, Wong, and Lightstone (2014)

results of it are used in the next step where the putative targets are docked to Nelfinavir using two docking tools: Surflex [208] and eHiTs [209]. These two steps form an MDE. A set of top ranked receptors undergo detailed docking and some of them are further analysed using MD simulations and MM/GBSA energy calculations. This step represents an AT. The results of the MD simulations are further processed when RMSD is calculated between superimposed structures, this additional step is another AT. They do not store the docking results or any other output (Figure 5.2).

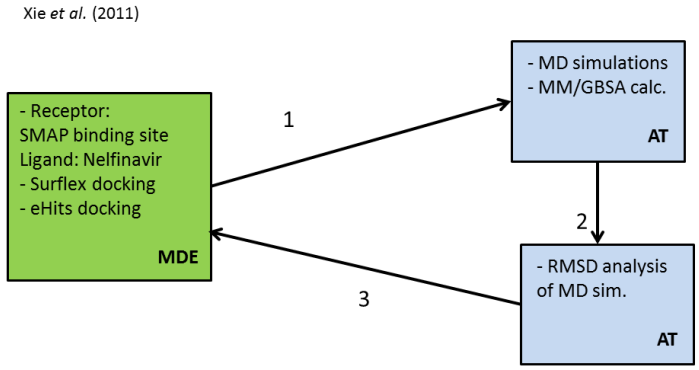

Figure 5.2: Basic diagram of Xie, et al. (2011)

MDE: Pre-processing using SMAP and docking using Surflex or eHiTs.

ATs: MD simulations and MM/GBSA energy calculations, and RMSD analysis.

Title: DOVIS 2.0.

Description: Jiang, et al. (2008) [118] present DOVIS 2.0, a system that uses AutoDock 4 to run large-scale VS on Linux clusters. They use Perl scripts to control the flow of events, compute energy grids, and link external scoring programs to DOVIS 2.0. A run

includes two sequential steps. Firstly, a directory with the needed parameter files (either generated by default or copied from an existing project) is created. Then, the energy grids required for AutoDock are computed before starting the parallel docking process on a cluster with a PBS or LSF scheduler. This represents an MDE element. The docked ligand-receptor complexes are scored and top-ranking results saved as final output. The storage system, a type of MDRR, seems to be rudimentary. Users can provide a wrapper script of a third-party scoring algorithm in order to rescore the docked ligands. After rescoring, a separate clustering algorithm is used to cluster the docked ligand poses based on this additional score. These two steps are two ATs (Figure 5.3).

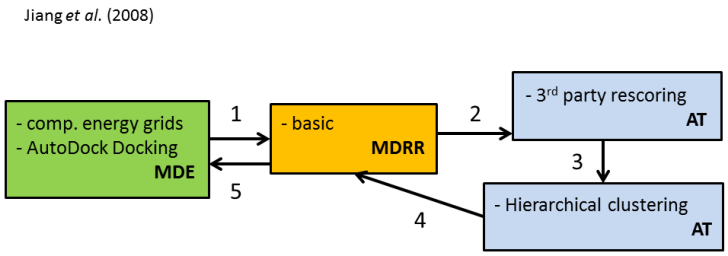

Figure 5.3: Basic diagram of Jiang, et al. (2008)

MDE: Pre-processing and docking using AutoDock 4.

MDRR: Rudimentary (not enough information).

ATs: 3^{rd} party rescoring, and hierarchical clustering.

Title: Generic framework for VS.

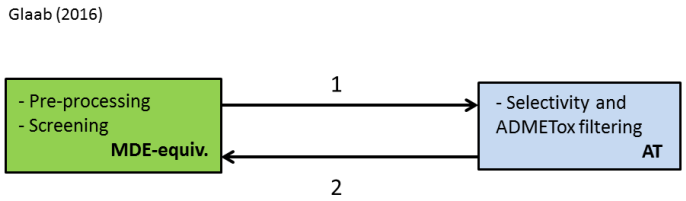

Figure 5.4: Basic diagram of Glaab (2016)

Description: Glaab (2016) [210] presents a review of current open-source programs that can be used in a VS pipeline. Based on it, he produces a generic framework for VS which includes the following elements: data collection, pre-processing, screening, and selectivity and Absorption, Distribution, Metabolism, and Excretion, Toxicity (ADMETox) filtering. The data collection element can be composed of external data sources used to acquire the input files for the VS simulation, such as wwPDB, or ZINC. The pre-processing step can

include quality control, structure pre-processing, or compound library pre-filtering. The third element, screening, can be viewed as receptor-based (e.g. docking with AutoDock Vina) or ligand-based (e.g. similarity search with Open Babel [211]). The pre-processing and screening steps together represent an MDE. An MDRR is not described in Glaab's framework. Instead, the fourth and last element in his framework, selectivity and ADME-Tox filtering, represent a broad group of ATs. It can contain reverse screening, ADMETox prediction, or an expert system for ADMETox filtering. Glaab only describes these elements and provides a Docker container with installed tools and a rudimentary script to run AutoDock Vina with example ligands and receptors (Figure 5.4).

MDE: Pre-processing and screening.

ATs: Selectivity and ADMETox filtering.

Title: VS pipeline to analyse the Neuraminidase of Influenza A and B virus N1.

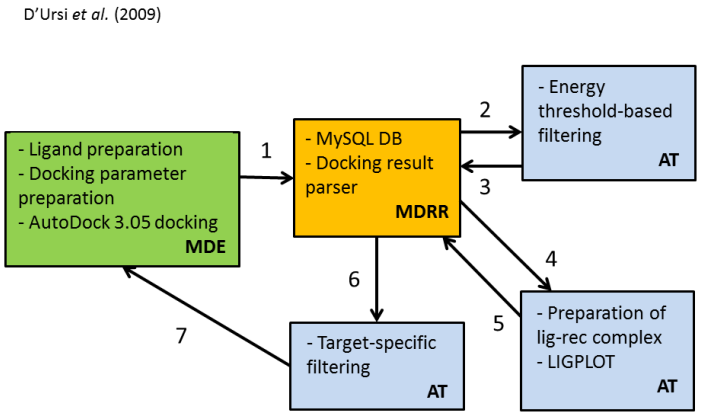

Figure 5.5: Basic diagram of D'Ursi, et al. (2009)

Description: D'Ursi, et al. (2009) [212] describe a VS simulation (with AutoDock as docking tool) using 3D structures of the influenza A and B virus N1 neuraminidase and ligand dataset from the DUD dataset [66]. They provide a semi-automated pipeline for VS and result processing which integrates the docking tools with analysis tools using Perl scripts. The pipeline begins with preparation of input files. It converts files into appropriate formats and automatically prepares the needed docking parameter files. Then, the pipeline runs the docking experiments on a PBS Linux cluster. These steps represent an MDE. When the docking is completed, the pipeline parses the results to find the best ligands. The backbone of their VS pipeline is a MySQL database where input and output from simulations and analysis tools are stored. The docking energy, docking cluster population, and atomic coordinates of the docking results are stored in the database, which is a type of an MDRR. In addition to this processing step, a tool is used to filter the ligands

with docking energy above a fixed threshold. The threshold-based filtering step represents an AT. A second tool is used to prepare the coordinates of the ligand-protein complex and analyse the interaction with LIGPLOT [213] which represents a second AT. The LIGPLOT output files are stored in the database and are accessible. Finally, a target-specific filter is applied in order to select compounds with specific patterns. This is a third AT. A small set of ligands are returned to the user as final results (Figure 5.5).

MDE: Preparation and docking with AutoDock.

MDRR: MySQL database.

ATs: Energy threshold-based filtering, LIGPLOT, and target-specific filtering.

Title: Docking using cloud computing on Microsoft Azure.

Description: Kiss, et al. (2014) [124] have developed a Windows-Azure-based cloud computing solution for running docking simulations (more details in Chapter 2). Their solution includes two scenarios with AutoDock 4 and a third with AutoDock Vina. They have developed a small self-contained .NET bundle to be installed on the user's computer, which connects to the Windows Azure Cloud where the docking is executed in parallel. This small desktop application enables users to upload the input files, configure the cloud instances and visualise the docking results. It represents the MDE. The results of the docking are stored on the cloud and can be downloaded to the user's computer using the small desktop application through a rudimentary version of an MDRR (Figure 5.6).

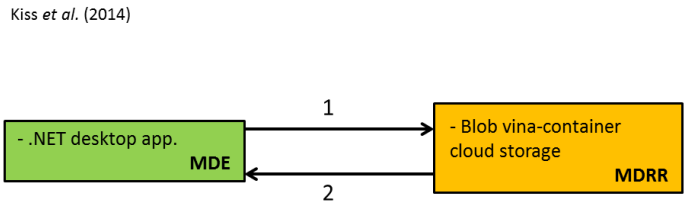

Figure 5.6: Basic diagram of Kiss, et al. (2014)

MDE: A .NET desktop application bundle.

MDRR: Cloud and local storage of docking results.

5.2.2 Workflow-based docking systems

Workflow-based docking systems prepare the docking input files, conduct the docking using a docking algorithm, and analyse the docking results with the help of scientific workflows.

The same format is used to describe 4 existing workflow-based docking systems.

Title: AutoDock Gateway based on WS-PGRADE/gUSE.

Description: Farkas, et al. (2015) [214] present an AutoDock gateway for docking in cloud systems. They use the WS-PGRADE/gUSE technology and create two workflows for AutoDock 4 and a third workflow for AutoDock Vina. Through a WS-PGRADE portal, users can provide input files required for the AutoGrid and AutoDock steps of the first workflow which runs AutoGrid. The second workflow assumes AutoGrid has been run by the users on their computer and requires them to upload the remaining input files only. The third workflow requires uploading the AutoDock Vina input files. The workflows can be run on a cloud using WS-PGRADE/gUSE and the CloudBroker platform. These steps represent an MDE. The results of the docking are stored on the gUSE server in the typical manner that WS-PGRADE/gUSE stores workflow results. Since only the creator can view their own workflow results and they cannot be used for an additional calculation directly, this represents a relatively simple type of an MDRR (Figure 5.7).

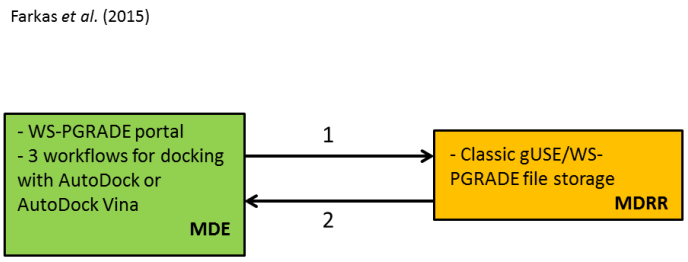

Figure 5.7: Basic diagram of Farkas, et al. (2015)

MDE: WS-PGRADE portal with 3 workflows.

MDRR: The classic WS-PGRADE/gUSE file storage.

Title: ProSim Gateway.

Description: Kiss, et al. (2010) [179] implemented the ProSim gateway, a gateway for docking and MD simulations based on WS-PGRADE/gUSE. The gateway includes a complex workflow consisting of four phases. Phase 1 includes receptor preparation steps such as solvation and charge neutralisation, energy and charge minimisation, and validation using the Molprobity [215] and GROMACS [104]. Analogous steps are used in Phase 2 to prepare the ligand. Phase 3 includes definition of grid space docking parameters and related steps required for docking using AutoDock. This phase produces ranked conformations by the lowest binding free energy. The user can visualise the results or they can be used by additional tools in Phase 4. The first three phases represent an MDE.

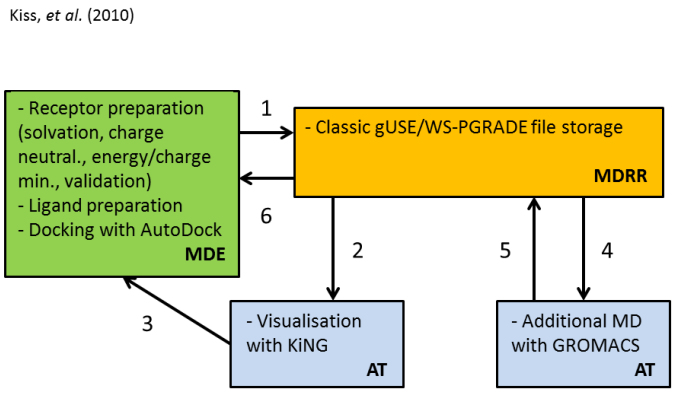

Figure 5.8: Basic diagram of Kiss, et al. (2010)

The visualisation is done using a separate portlet and the KiNG visualisation tool [216] which can be viewed as a separate AT. The fourth and final phase refines the ligand-receptor complex using MD simulations which represents an AT. The traditional WS-PGRADE/gUSE storage system which stores the workflows, input and output results is utilised in ProSim. This represents a basic MDRR element (Figure 5.8).

MDE: Preparation of receptor and ligand, docking with AutoDock.

MDRR: The classic WS-PGRADE/gUSE file storage.

ATs: Visualisation with KiNG and additional MD with GROMACS.

Title: AMC Docking Gateway.

Description: Jaghoori, et al. (2015) [116] describe the Docking Gateway at AMC, University of Amsterdam. The Docking Gateway uses AutoDock Vina as the docking engine and includes a front-end where the user can upload the input items: receptor, ligands and AutoDock Vina configuration file. The status of submitted jobs can be monitored from the front-end, as well as a visualisation of the docking results of completed jobs. This represents a type of MDE. Basic provenance information is stored about the processing actions including information about the users, their data and the applications (different implementations of AutoDock Vina based on three infrastructures: gUSE, DIRAC [217], or Hadoop [123]). The management and maintenance of this information is done by a component called Processing Manager. This can be viewed as an MDRR. The input files attached by the user are split and parallelised, the docking tool is run, and then they are merged. Apart from monitoring and visualisation, no additional tools interact with the docking results (Figure 5.9).

MDE: Uploading input files for AutoDock Vina.

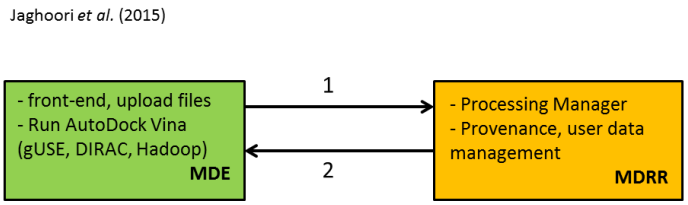

Figure 5.9: Basic diagram of Jaghoori, et al. (2015)

MDRR: Management of provenance and user data.

Title: MoSGrid Portal.

Description: Krüger, et al. (2014) [146] have developed MoSGrid, a portal-based science gateway that uses WS-PGRADE/gUSE technology to run molecular simulations from three domains, including docking, on a computing grid (more details about MoSGrid are provided in Chapter 2). The docking section of MoSGrid enables users to employ the CADDSuite [147], AutoDock, and FlexX [83] docking tools to run docking experiments through a WS-PGRADE portal. This forms an MDE. MoSGrid contains elaborate data storage components which store raw, preliminary, and result data in the distributed file system XtreemFS [218]. These components utilise the custom-made MSML [143] description language for transfers, conversions and analysis of data files. MoSGrid includes a simulation repository based on gUSE and the UNICORE [219] Metadata Service (which is based on Apache Lucene). Within the simulation repository, the workflow files are converted to MSML and extended with docking results and other metadata which is then used for indexing in order to enable searching. The structures of the data files are stored in a repository where they are converted from the original PDB format to MSML. These components represent a robust MDRR. The workflows can be monitored as in all WS-PGRADE/gUSE systems. On top of that, MoSGrid provides a visualisation tool (ChemDoodle [220] for 3D structures and Dygraphs for 2D plots). This represents an AT. The docking workflow includes four steps: target preparation, ligand preparation, docking, and rescoring. The rescoring step represents an additional calculation of the docking results, so it can be viewed a separate AT (Figure 5.10).

MDE: Preparing target and ligand, docking with CADDSuite, AutoDock, or FlexX.

MDRR: MSML format stored in XtreemFS, as well as JSON for indexing with Lucene.

ATs: ChemDoodle or Dygraphs visualisation, and rescoring the docking.

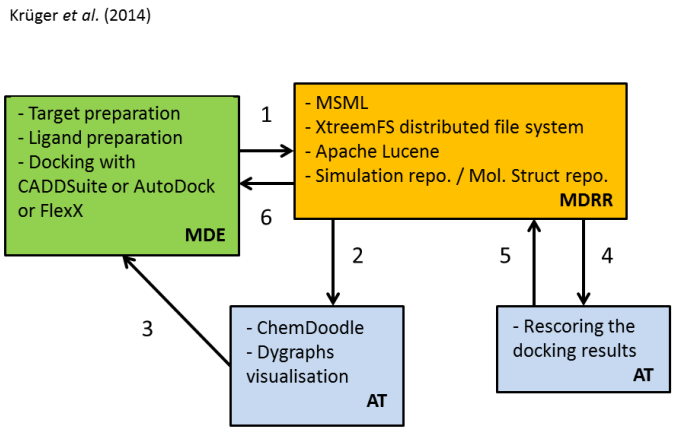

Figure 5.10: Basic diagram of Krüger, et al. (2014)

5.2.3 Docking-equivalent systems

Docking-equivalent systems do not use a docking per se, but rather an equivalent type of bioinformatics tool. Their elements can be classified into groups equivalent to the framework's element types. A total of 4 systems are described.

Title: PoLi.

Description: Roy, Srinivasan, and Skolnick (2015) [221] describe the pipeline PoLi. PoLi includes several pre- and post-LBVS calculations. At the start of the pipeline the user uploads the 3D structure of the receptor. If the structure is unknown the user may upload the sequence and the TASSER-VMT [222] tool will be used to model the 3D structure. In the next step, two different approaches are used to detect the ligand binding site: structural alignment of the target receptor and other proteins in the PDB using TM-align [25], and detecting pockets on the target protein using ConCavity [223]. The predicted pockets are then compared to known ligand binding sites using the tool APoc [224]. The results are template ligands which are then pruned before running LBVS simulations. Two methods are used in the LBVS step, the first is the shape-based ligand similarity tool LIGSIFT, and the second is a fingerprint-based calculation using Open Babel [211]. This step does not use docking simulations, but it is conceptually analogous, so it can be defined as an MDE-equivalent. The definition of AT assumes that some tool is used to process the docking results in addition to any pre-docking tools. Therefore, in this analogous example, all the pre-LBVS steps would be categorised together with the LBVS step into one MDE-equivalent element. An additional step to analyse the LBVS results is used in PoLi, whereby a fusion technique to combine the results of the two LBVS methods has been defined. This represents an example of an AT (Figure 5.11).

MDE: Preparations for LIGSIFT and OpenBabel LBVS.

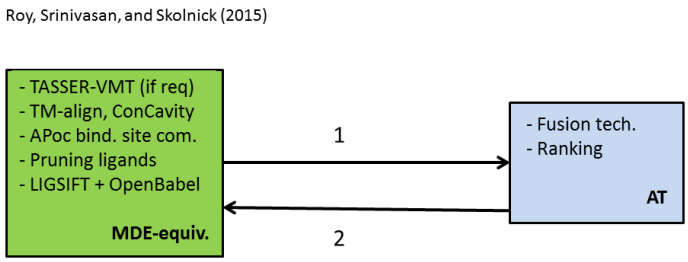

Figure 5.11: Basic diagram of Roy, Srinivasan, and Skolnick. (2015)

ATs: A technique for fusion of results, and ranking.

Title: WeNMR Portals.

Description: Wassenaar, et al. (2012) [225] describe WeNMR a large portal-based system which contains a separate portal for 19 different domains and uses grid computing for the execution of programs. One of the domains is protein-protein docking using HADDOCK [226], which can be viewed as analogous to protein-ligand docking. HADDOCK consists of an initial rigid-body docking followed by a flexible refinement and scoring process. The users have 4 types of interfaces to choose from, based on their expertise with HADDOCK. This represents an element equivalent to the MDE. The progress of the docking can be followed on the portal's website. Once completed, the results of the docking are stored for a limited time in a file-system-based storage facility and can be viewed online or downloaded. This is an MDRR-equivalent element. WeNMR includes two portals for MD simulations using AMBER [121] or GROMACS. The AMBER portal requires the user to go through 4 pre-MD steps before starting the simulation: protein optimisation, setting NMR restraints, setting MD parameters, naming and submitting the calculation. The GROMACS process starts with force-field-specific protein topology, then includes solvation and equilibration. WeNMR includes an interface where users can upload pre-equilibrated proteins before the GROMACS MD simulation is run. Therefore, the AMBER and GROMACS portals can be described to have an MDE-equivalent element (Figure 5.12). When a job finishes several post-processing steps extract statistics from the result files for viewing online, then clean and store the job files for download.

MDE: Steps to prepare and run HADDOCK, AMBER, or GROMACS.

MDRR: File-system-based shared storage.

Title: GridMACS.

Description: Chia et al. (2010) [103] present GridMACS a grid-computing-based portal

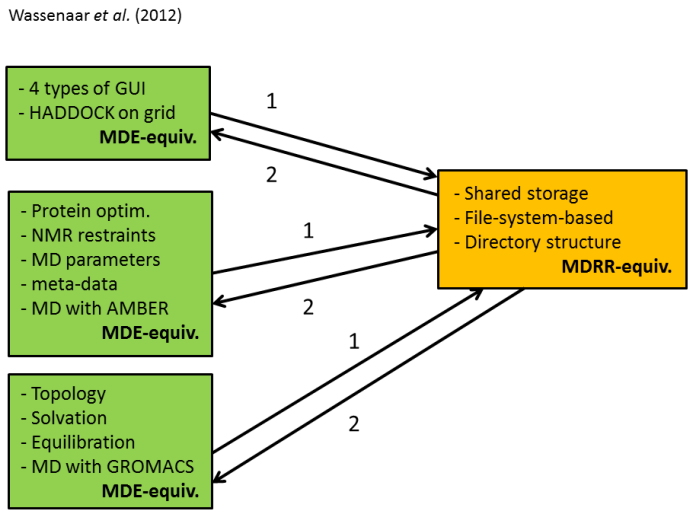

Figure 5.12: Basic diagram of Wassenaar, et al. (2012)

for running GROMACS MD simulations. Users can upload the required input files and the GROMACS executable file and submit a job to the grid. This is an equivalent element to the MDE. When the job is completed, its output is sent to a component called File Manager. This element includes a file system where the results of the MD simulations are stored. Users can interact with the files, for instance they can download, upload, rename, or delete files. This is equivalent to an MDRR (Figure 5.13).

MDE: Uploading files for MD simulations with GROMACS.

MDRR: File-system-based storage.

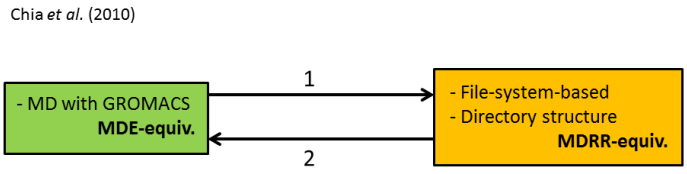

Figure 5.13: Basic diagram of Chia, et al. (2010)

Title: iPortal (the new version of the Swiss Grid Proteomics Portal).

Description: Kunszt, et al. (2015) [227] have developed iPortal which uses proteomics data analysis methods. Three WS-PGRADE workflows called "search", "quantification", and "SWATH" are included. The workflows can be parametrised before submission. The quantification workflow makes use of results of the search workflow and additional information from historical reference data. In this respect, the search workflow can be viewed as the main workflow and as such equivalent to an MDE element. An openBIS-based [228]

component manages all the data which is stored in a directory structure and registered in an openBIS database. The files can be accessed over the web interface or retrieved using the openBIS API. Metadata, links between the data and metadata, or tracking data provenance is provided by the openBIS element. This is equivalent to the MDRR element. The results of the search workflow are stored before they (along with other information) are used by the quantification workflow. Thus, the quantification workflow can be viewed as an AT. The search workflow requires access to reference data, usually a subset of the external and publicly available database UniProt [229]. The data are loaded into the openBIS storage system via an element called BioDB. BioDB regularly downloads data from UniProt providing versioning information and the required data enrichments. This element is equivalent to the ADS. An analogous element called PersonalDB allows users to upload their custom data source, which, if used, represents another ADS (Figure 5.14).

MDE: The preparation and conducting of the "search" workflow would be equivalent.

MDRR: Meta-data, provenance and results stored in an openBIS-based system.

AT: The "quantification" workflow.

ADSs: BioDB and PersonalDB.

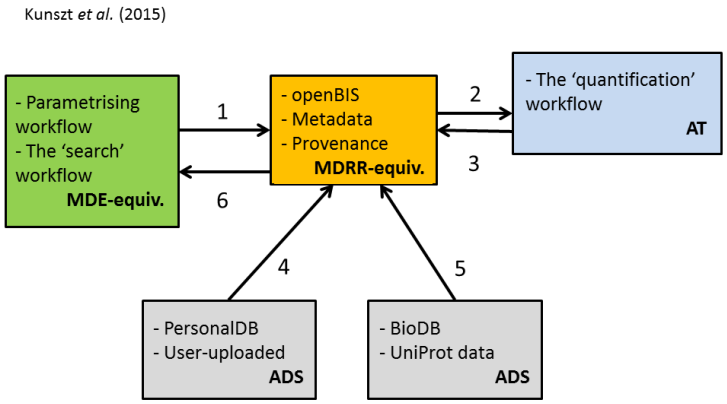

Figure 5.14: Basic diagram of Kunszt, et al. (2015)

This section provided a literature review of 14 existing systems divided into VS pipelines, workflow-based docking systems, and docking-equivalent systems. For instance, D'Ursi *et al.* (2009) [212] have created a VS pipeline. After preparing the input files and conducting the docking using AutoDock, they store the results in a MySQL database, and provide methods for target-specific filtering, filtering based on an energy threshold, and creating schematic diagrams of the ligand-receptor complex. In a workflow-based docking system, Kiss *et al.* (2010) [179] prepare input files, conduct docking, store docking results as part of the workflows, and further analyse the docking results by running MD simulations

or visualising them. An example of a docking-equivalent system includes Kunszt, *et al.* (2015) [227] which use different proteomics data analysis methods, store the results in a file system, and retrieve data from an external data source (UniProt [229]).

5.3 Conclusion

All of the 14 existing systems outlined in this chapter can be described using the high-level diagram of the framework. Including the 5 novel scenarios obtained from the primary research, this shows that a total of 19 systems can or could have been developed using the framework. Furthermore, this literature review shows that even when it is not obvious, such as with docking-equivalent systems, the idea of the framework could have been used. This is enough evidence to posit that any novel system that uses stored previous docking results can be described using the framework. To prove this, three of the 5 novel scenarios will be described with the more detailed views of the framework which will be introduced in Chapter 6. Chapter 8 will describe how these three scenarios have been implemented and tested.

Chapter 6

Low-Level Description of Element Types and Interfaces

The low-level diagram addresses the lack of an abstract conceptual framework in more detail. The framework, designed for software systems that use previous docking results, is independent of the programming language, toolset, or paradigm used. Using the detailed diagram of the framework can prevent the development of a tool from scratch if an existing tool can be reused. A tool can be described abstractly (drawn as a component of the diagram) and compared to other abstract descriptions of existing tools. Furthermore, if a description of another existing tool does not exist and a software engineer needs to determine whether the tool will fit the framework, it can be compared to an abstract description of an element type. This chapter shows the low-level view of the framework in the form of the diagrammatic, textual and formal descriptions of element types and interfaces.

6.1 Diagrammatic description of the framework

One should be able to represent any potential specific scenario that would use the framework, with this type of low-level detailed diagram. This diagram is a generic model of a system that uses a docking result repository, showing all element types and all possible interfaces between them. It is based on the Unified Modelling Language (UML [230]) Component diagram. The element types are drawn as components and the interfaces between them are the typical "provided" and "required" interface connections. It also features arrows showing the direction of the flow of data in the particular interface. The element types MDE, MDRR, AT, ADS, and DM were described in detail in Section 4.3.3.

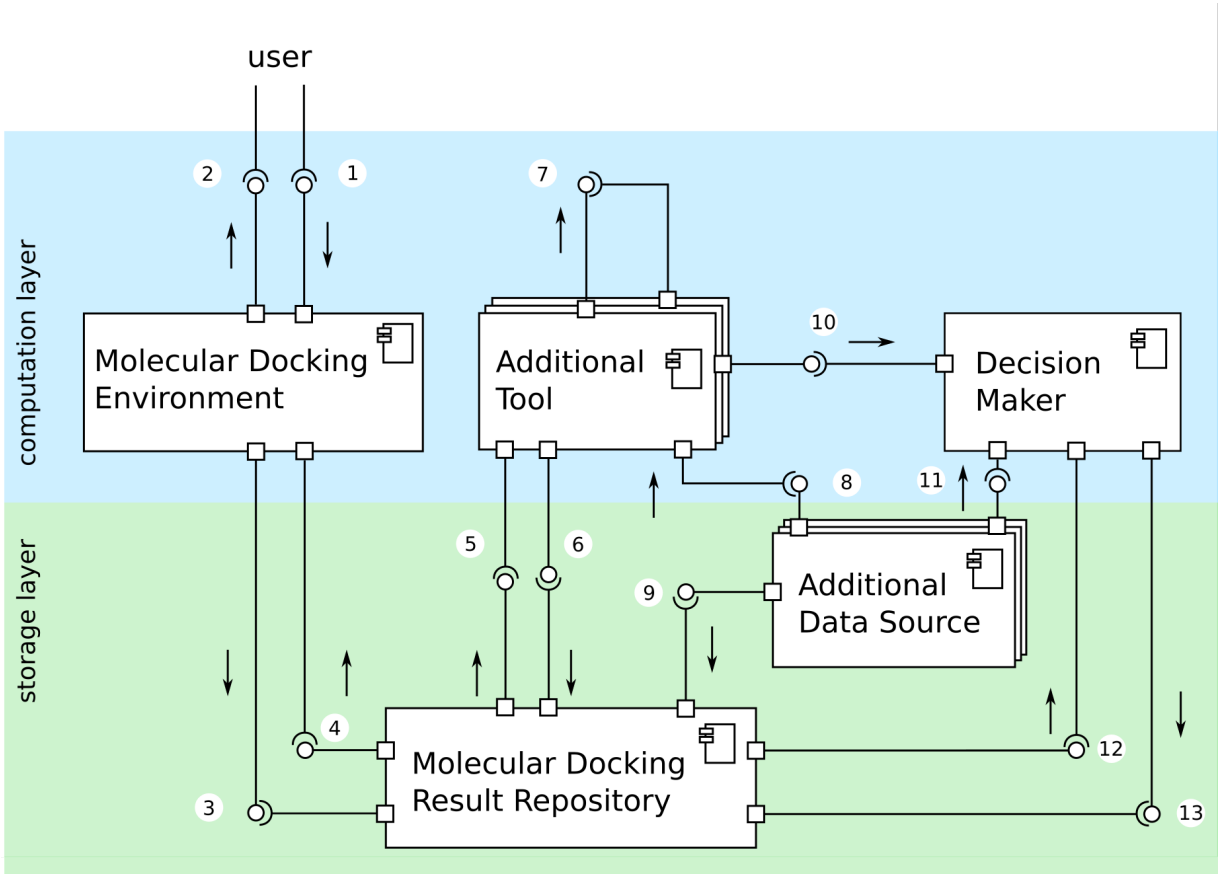

Figure 6.1: The diagram of the framework.

The framework has 13 interfaces between these element types (Figure 6.1):

1. user → MDE, provided by the MDE (since the user is not a true component): allows the user to upload the docking input or additional user input values needed by another element.
2. MDE → user, provided by the MDE: displays the result of the docking and other results from the MDRR to the user.
3. MDE → MDRR, provided by the MDE: allows the MDE to send docking results and other additional data that may be required.
4. MDRR → MDE, provided by the MDRR: allows the MDRR to send results of the analysis to the MDE.
5. MDRR → AT, provided by the MDRR: enables sending the appropriate input data to the AT.
6. AT → MDRR, provided by the AT: allows the AT to send results of the execution to the MDRR, in order to store them and keep track of the progress.
7. AT → AT, provided by the AT: allows one AT to send its results, or other required data, to another AT.

8. ADS → AT, provided by the ADS: enables querying the ADS for data, by the AT.
9. ADS → MDRR, provided by the ADS: querying the ADS for data, by the MDRR.
10. AT → DM, provided by the AT: allows an AT to send the results to the DM.
11. ADS → DM, provided by the ADS: enables querying the ADS for data, by the DM.
12. MDRR → DM, provided by the MDRR: allows the MDRR to send data to the DM.
13. DM → MDRR, provided by the DM: enables the DM to send results to the MDRR.

6.2 Textual description of element types and interfaces

MDE A tool that takes descriptions of molecules as input, runs molecular docking, and outputs the results. It can be a bundle of tools, which pre-process the description files, run the calculation, and process the results. It could contain a set of shell scripts, a workflow, or a set of workflows. The computing infrastructure it uses is completely independent of the MDE (Figure 6.2).

MDE interfaces

1. The MDE requires users to send files describing the ligands, receptors and the configuration file. The user sends the input files to the MDE. For example, Raccoon2 is an MDE for VS simulations, and it has a GUI that lets users upload a receptor, ligands and a config file. The user may send other data that is needed by another element (the user is not a software component, so this interface cannot be provided by the user).
2. The MDE provides an interface which displays the results back to the user.
3. The MDE provides an interface in order to deposit the results into an MDRR. All needed information should be sent including input and output (result) files.
4. The MDE needs to receive any analysis data or decision made directly from the MDRR. This requires an interface at the MDRR to send this data over.

MDRR A storage system where the molecular docking results are stored. It should also store meta-data so that the simulation from the MDE is reproducible. This can be a database with certain textual data as well as pointers to the path of files stored in a file system, or it could be a document-based NoSQL database where all the files are stored inside the database, or something similar (Figure 6.3).

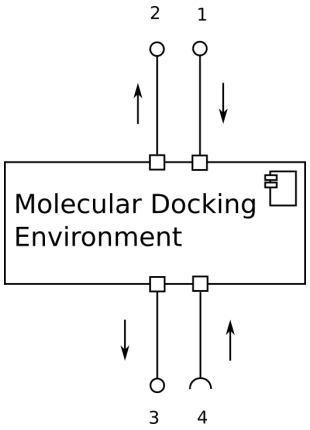

Figure 6.2: Diagram of the MDE.

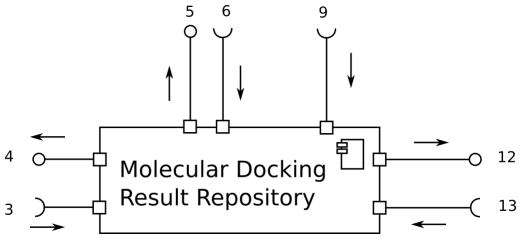

Figure 6.3: Diagram of the MDRR.

MDRR interfaces

3. The MDRR requires an interface to read the docking results and other data uploaded.
4. It provides an interface to send the result of the analysis to the MDE.
5. It provides an interface to send stored docking results data, in the needed format, to an AT.
6. It requires an interface to receive the AT results and keep a record of them.
9. The MDRR requires an interface in order to receive data from an ADS.
12. It provides an interface for sending stored docking results data in the needed format, to the DM.
13. Finally, it requires an interface to receive the AT results and keep a record of them.

AT The element type AT can be a tool that takes previous docking results from the MDRR as input, and runs another computation relevant to the particular scenario. It could take input data from a user sent through the MDE – MDRR, or from an ADS. It does not conduct docking simulations, but uses previous docking results or the input files for previous docking simulations. For instance, two receptors used in two docking experiments can be compared with a structural alignment tool. The AT can pass the results onto another AT, a DM, and it may send them to the MDRR (Figure 6.4).

AT interfaces

5. An AT (you can have multiple ATs) requires an interface that enables the MDRR to send stored docking results data so it can use them as input.

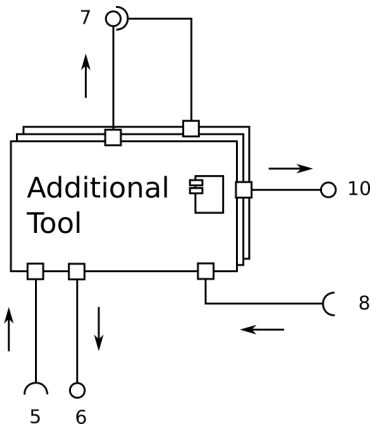

Figure 6.4: Diagram of the AT.

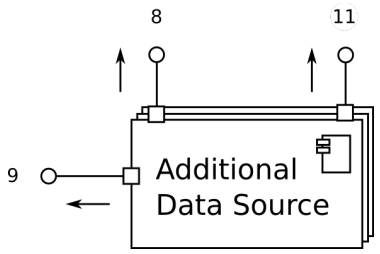

Figure 6.5: Diagram of the ADS.

6. An AT provides an interface that enables it to send its results to the MDRR to keep track of them.
7. An AT may provide an interface to send its results to another AT.
8. An AT may require an interface in order to obtain data from an ADS.
10. An AT provides an interface in order to send its results to the DM.

ADS The element type ADS represents a tool, such as a database, that stores relevant data. An ADS does not store the docking results, but other type of data that is additionally required in order to analyse previous docking results. It stores data that needs to be accessed by an AT, DM, or the MDRR. For instance, molecular properties about ligands can be read from an existing database. A scenario could read data from an external database, or include a copy of the database as part of the system (Figure 6.5).

ADS interfaces

8. An AT may need to use data stored in an ADS. The ADS should provide access to the data it stores, so that the AT can access it from its code.
9. It needs to provide access to its data for the MDRR as well.
11. Similarly, it needs to provide access to its data for the DM.

DM A tool (or bundle of tools) which make a decision based on results from an MDE and/or an AT. It gets the results directly from an AT, and can get more information from the MDRR. It may need to obtain additional information from the user (Figure 6.6).

DM interfaces

10. The DM requires an interface to receive results from one or more ATs.
11. The DM may require an interface defined at the ADS in order to obtain data from an ADS.
12. The DM requires an interface to receive previous docking results and other data from the MDRR.
13. The DM provides an interface to send the decision made to the MDRR.

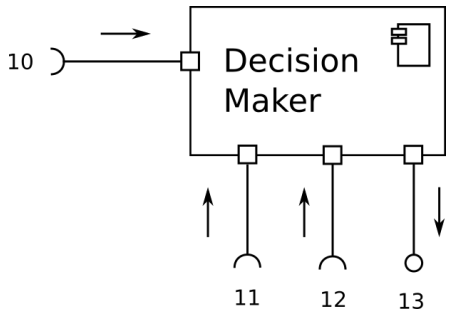

Figure 6.6: Diagram of the DM.

This concludes the low-level description of the framework with a detailed diagram and textual description of the element types and interfaces.

6.3 Formal description of element types and interfaces

To provide more objective means to compare an asbtract view of an existing tool with an element type, this section provides a formal description using Z notation (the rationale for using Z was outlined in Chapter 2). Using formal methods in the field of molecular docking is limited to research efforts such as [231], which does not utilise the popular *Z notation*. The formal description of element types and their interfaces has been written in CZT Eclipse [232], which automatically checks that the code conforms to the Z syntax. The full formal description of the framework is provided in Appendix B. It begins with a freetype and set definitions, assuming that there is a set *CHAR* which represents allowed characters. Regardless of the format that the input and output files use, they can be viewed as containing strings of characters. Multiple files are modelled using the set operator ($\mathbb{P}$). A mapping of the tuple ligand-receptor-config-date to a docking result is modelled as a previous docking result, *PREVIOUS_RESULT* (page 162).

6.3.1 Element types

MDE An MDE enables the execution of a docking simulation. A docking process requires a ligand and a receptor as input, and produces a docking result as output. Depending on the docking tool or use case, it may or may not require a configuration file (or configuration parameters). This is modelled by *dockingWithoutConfig*, *dockingWithConfig*, *Docking*, and *MolecularDockingEnvironment* in Appendix B, pages 162 - 163. An excerpt of the formal description showing the segment related to the MDE is shown in Figure 6.7.

$$dockingWithoutConfig : (LIGAND \times RECEPTOR) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists res : RESULT \bullet dockingWithoutConfig(l, r) = res$$

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking

$$ligand? : LIGAND$$
$$receptor? : RECEPTOR$$
$$config? : CONFIG$$
$$result! : RESULT$$

$$config? = \varnothing \land result! = dockingWithoutConfig(ligand?, receptor?) \lor$$
$$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$$

MolecularDockingEnvironment

$$ligands? : LIGANDS$$
$$receptors? : RECEPTORS$$
$$config? : CONFIG$$
$$results! : RESULTS$$
$$date! : DATE$$

$$\exists ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking$$

ViewMolecularDockingResults

$$\Xi MolecularDockingEnvironment$$

$$results! \neq \varnothing$$

Figure 6.7: Excerpt of the Z notation describing the MDE element type.

MDRR The MDRR should store data about the relation between a ligand-receptor-config-date tuple, and a docking result. It may include the decision made by a DM. The model is a minimal MDRR, acknowledging that some scenarios may require storing additional data such as author or version of docking tool used. They could be defined in a similar way to ligand or receptor, and included in the definition of *repository*. This would be reflected in all the interfaces to and from the MDRR. However, to compare a formal description of an existing tool to this abstract description of an MDRR, only the minimal data stored is required, as modelled in *MolecularDockingResultRepository*, Appendix B, page 164.

AT The AT is perhaps the most generic element type. The only restriction on the calculation it provides is that it is not another docking simulation. Different sub-types of ATs can be defined based on the type of input. Thus, an AT can use previous docking results (from an MDRR), data source information (from an ADS), additional tool result (from another AT), or a combination of them (which may also include user input). This is modelled by the axiomatic definitions in Appendix B, pages 165 - 168. The fact that an AT is defined by one of these types is shown in the schema *AdditionalTool*.

ADS The ADS is also modelled in a very generic manner. It is a database that stores data which is relevant for the system. This is modelled as a relation between a generic data source input and a resulting data source information. The very simple *Additional-DataSource* in Appendix B, page 169 shows this.

DM The purpose of the DM is to summarise the results it has received from ATs, the MDRR, or the user. The way that sub-types of DMs have been identified is similar to the identification of sub-types of ATs. Based on the combinations of inputs, there are several sub-types of DMs as shown in Appendix B, pages 169 - 171. The fact that a DM is defined by one of these types is shown in the schema *DecisionMaker*.

6.3.2 Interfaces

Interface 1: User, MDE This interface is modelled by specifying the user-provided text files as input variables (using the suffix "*?*"), Appendix B, page 163.

Interface 2: MDE, User This interface is modelled by a schema showing that the docking results can be viewed by the user, as long as they exist and they are output variables (suffix "*!*"), Appendix B, page 163.

Interface 3: MDE, MDRR An MDE can insert one or more docking results, as modelled by two schemas in Appendix B, page 164. The Δ operator signifies that there will be a change, which is detailed by the override operator ($\oplus$). A new item will be inserted, unless the ligand-receptor-config-date tuple is the same as an existing one, in which case the MDRR will be updated.

Interface 4: MDRR, MDE Interfaces 4, 5, and 12 represent the selection of data from the MDRR into the MDE, AT, or DM respectively. The repository is modelled as a relation, so the appropriate domain ($\lhd$) and range restriction ($\rhd$) operators are used. The former is used to select a tuple based on the left-hand side of the relation (when the ligand, receptor, config, or date is known), and the latter based on the right-hand side (when the docking result is known). A total of 32 different combinations of selection types are outlined in the schema *SelectMolecularDockingResults*, Appendix B, page 165. The same schema models interfaces 4, 5, and 12.

Interface 5: MDRR, AT Please see description of Interface 4. Interface 5 is further described in *AdditionalTool* by the input variables *previousDockingResults* and *userInput* (which would be passed via the MDRR).

Interface 6: AT, MDRR This interface can be modelled similarly to the way that Interface 3 models the MDE inserting docking results into the MDRR. In order to do this, the formal description of the MDRR would need to include results from an AT.

Interface 7: AT, AT Additional tool results of another AT are defined as an input variable in the *AdditionalTool* schema showing that an AT can receive results of another AT. This action is described in more details in *ReadAnotherAdditionalToolResults* (Appendix B, pages 168 - 169).

Interface 8: ADS, AT An AT can use data from the ADS by selecting it accordingly. This action is modelled by the schema *SelectAdditionalDataInfo* which shows that data from the ADS can be selected (Appendix B, page 169). The right-hand side of the repository (data source info) is selected based on the value of the left-hand side (data source input). The domain restriction ($\lhd$) is used to select the items stored in the ADS. The range function, *ran()* is used to obtain the right-hand side values for the selected items.

Interface 9: ADS, MDRR This interface can be modelled similarly to the way that Interface 3 models the MDE inserting docking results into the MDRR. In order to do this, the formal description of the MDRR would need to explicitly include results from an ADS. The schema *SelectAdditionalDataInfo* shows that data from the ADS can be selected (Appendix B, page 169).

Interface 10: AT, DM The fact that the *additionalToolResult* is defined as an input variable in *DecisionMaker* (Appendix B, 171) is sufficient to model an interface between an AT and DM. If a sub-type of DM requires results from an AT as input, they can be received.

Interface 11: ADS, DM Please see Interface 8. The same schema is used to model the interface that the DM uses to select data from the ADS.

Interface 12: MDRR, DM Please see description of Interface 4. Interface 12 is further described by the fact that the *userInput* variable in *DecisionMaker* is an input variable (which would be passed through the MDRR).

Interface 13: DM, MDRR A new decision from the DM can be inserted into the MDRR, or an existing one can be updated. This is modelled with the help of the override operator ($\oplus$) in *InsertUpdateDecisionRepository*, Appendix B, page 164.

6.4 Conclusion

This chapter proposed an abstract conceptual framework which is independent of the programming language, toolset, or paradigm used by a software system. The framework can be used to describe systems that use previous molecular docking results. It provides three main functionalities. Firstly, a scenario can be described using the basic diagram of the framework in order to determine whether it could be implemented using the framework. Secondly, the use of the framework will create a library of abstract descriptions of existing tools. When seeking an existing tool to use in a system, the abstract description of the element type can be compared to the library to find a candidate existing tool. Thirdly, it includes abstract descriptions of generic element types. An existing tool can be described in the same format and compared to the appropriate element type, to find out if it can be used in an implementation of a system. The three uses of the framework are further described as the techniques of the methodology in Chapter 7.

The research methodology that led to the construction of the framework was outlined in Chapter 4. This included two types of interviews (primary research), and a literature review of existing systems that store docking results (secondary research - Chapter 5). It resulted in a high-level view of the framework which can be used in 5 novel scenarios and could have been used in 14 existing systems from the literature. Furthermore, a low-level view composed of a diagrammatic, textual, and formal description of the generic element types and interfaces of the framework was presented in this chapter. Chapter 7 proposes a specific software development methodology for using the framework. Chapter 8 evaluates the benefits of the framework and methodology, by using them to develop prototype implementations of three scenarios.

Chapter 7

Methodology for Developing Systems that Use Docking Results

7.1 Introduction

The definition of the term "methodology" which is used in this chapter is: "A series of related methods or techniques" (as explained in [233]). Such a methodology can be used for establishing practices, team rules and conventions. It should be useful when introducing new people to the process, substituting people, or delineating responsibilities. With this definition in mind, a description of a methodology for developing software systems that use docking results can be created. The aim of the methodology is to complement the framework (Chapters 4, 5 and 6) by showing how one can use the framework. The methodology should clearly state the required roles of the people involved, and the specific sub-projects for which they need to collaborate.

Creating the diagrammatic, textual and formal descriptions of elements and interfaces for a given scenario will add non-negligible amount of documentation to a methodology. If the methodology is bulky, complex, and heavy, then using the framework may be too difficult to manage. One way to tackle this issue is to create an agile methodology, which will be light-weight even once the abstract descriptions of the elements and interfaces have been included.

In software engineering, agile methodologies assume that customers are actively involved. In the case of software systems that use previous docking results, the life scientists who are the end-users, should be actively involved. Agile methodologies focus on delivering working software on short intervals. The methodology described in this chapter includes several design and planning steps prior to coding, in order to ensure reusability of domain-

specific elements, and an easier development of multiple scenarios.

Several authors have been writing about agile methodologies since the inception of the term in the early 2000s, when some of most influential authors created the Agile Manifesto [234]. This thesis focuses on the work of Alistair Cockburn, a co-creator of the Agile Manifesto. His work was chosen because it is based on a large number of interviews that he conducted with software project teams. This has resulted in, among others, the Crystal family of methodologies [235], and the main source for creating the methodology in this chapter [233].

In [233], he notes that the physical size of the methodology should be kept small by: providing examples of work products, removing the technique guides (instead of describing the techniques in detail, simply naming the recommended techniques along with any key literature), and organising the text by role. The so-called Role-Deliverable-Milestone pictorial view [233] can be used to organise the methodology by role. A version of this pictorial view is used to describe the methodology in the remainder of this chapter.

7.2 Role-Deliverable-Milestone diagram

The Role-Deliverable-Milestone pictorial view of a methodology has been proposed as a method of minimising methodology bulk, while providing sufficient information for each role. The central parts of the methodology presented here are the high-level (Figure 7.1) and low-level (Figure 7.2) Role-Deliverable-Milestone diagrams.

Roles The roles describe the different types of people that should be involved in the development team. In the methodology proposed in this thesis, they are: Life Scientist, Bioinformatician, Software Developer, Modeller, and IT Infrastructure Administrator.

Deliverables The deliverables are work products that need to be constructed at various points in the development process. In this methodology they are: Diagrammatic Description, Textual Description, Formal Description, and Final System Code. The code for the final software system should be divided into the five element types of the framework: MDE, MDRR, ATs, ADSs, and DM.

Milestones Milestones mark an important event. A scenario that uses the framework would have the following milestones:

M1 Diagram start.

M2 Diagram ready for review.

M3 Diagram ready.

M4 Textual description start.

M5 Textual description ready for review.

M6 Textual description ready.

M7 Formal description start.

M8 Formal description ready for review.

M9 Formal description ready.

M10 Software start development.

- M10.1 MDE start development.
- M10.2 MDRR start development.
- M10.3 ATs start development.
- M10.4 ADS start development.
- M10.5 DM start development.

M11 Software developed.

- M11.1 MDE developed.
- M11.2 MDRR developed.
- M11.3 ATs developed.
- M11.4 ADS developed.
- M11.5 DM developed.

M12 Software installed/deployed.

- M12.1 MDE installed/deployed.
- M12.2 MDRR installed/deployed.
- M12.3 ATs installed/deployed.
- M12.4 ADS installed/deployed.
- M12.5 DM installed/deployed.

M13 Software tested and ready.

- M13.1 MDE tested and ready.
- M13.2 MDRR tested and ready.
- M13.3 ATs tested and ready.
- M13.4 ADS tested and ready.
- M13.5 DM tested and ready.

7.2.1 High-level diagram

The high-level diagram (Figure 7.1) shows which roles should collaborate to create a particular product. Milestones M1, M2, and M3 should be reached when creating the deliverable: Diagram (diagrammatic description of the scenario, as specified in the framework). In the first iteration, a basic high-level diagram of the scenario will be created. M1 and M2 should be done jointly by the Life Scientist and Modeller. This can be read from the full line next to Diagram, between M1 and M2, which is shown in the first box displaying the tasks of the person with role Life Scientist, and the third box showing tasks for the Modeller. The white circle signifies that the deliverable Diagram should be started by these roles.

When milestone M2 is reached, the diagram is ready for review. At this point, the Bioinformatician and Modeller can use the abstract basic diagram of the scenario to assess whether the scenario can be implemented using the framework. The diagram of the scenario can be compared to the abstract basic diagram of the framework (Figure 4.2). This is the first added value of using the framework and this methodology. Concluding that

the scenario fits the framework will finish the first phase of the Diagram deliverable. In a second phase a detailed diagram of the scenario (derived from the detailed diagram of the framework in Figure 6.1) can be created. The creation of this diagram finishes the Diagram deliverable, indicated by the full black circle. However, the diagrams can be revised at key points in the development, as shown by the dotted lines, a component which has been added in order to clarify Cockburn's pictorial view. The Diagram deliverable is not completed yet, and can be altered when creating the Textual Description and Formal Description of elements and interfaces. Here, a textual list of interfaces, and a formal description should be created for each element. There are two main uses of these abstract descriptions.

If the development team searches for an existing tool that could be used, the abstract description of the element can be compared to a library of already implemented elements in other scenarios. This can identify a tool that can be reused in order to avoid creating an element from scratch.

If the team has an existing tool in mind, the team can check whether it can be used in the scenario. Once a diagrammatic, textual, and formal description of a proposed existing tool and its interfaces have been created, they can be compared to the abstract descriptions of the appropriate element type of the framework. This can show whether the existing tool can be used as an element in the implementation of a scenario.

In the high level Role-Deliverable-Milestone diagram, the coding section has an asterisk (*) because there is a more detailed description available. The repetition of M11 and M13 indicate the iterative coding process, during which, the diagrammatic, textual, or formal descriptions of the scenario may be revised (as shown by the dotted lines).

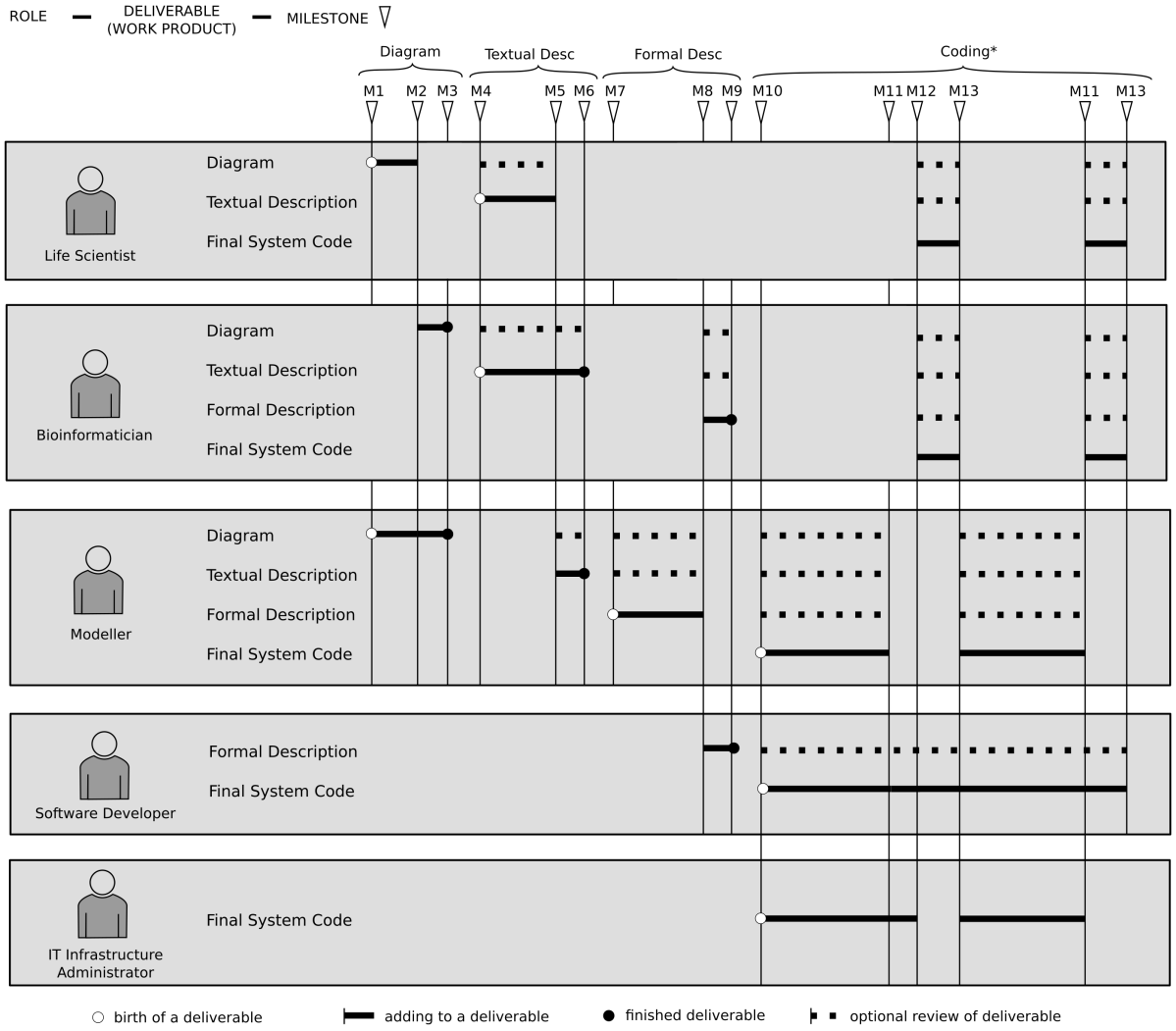

Figure 7.1: Role-Deliverable-Milestone diagram (high level).

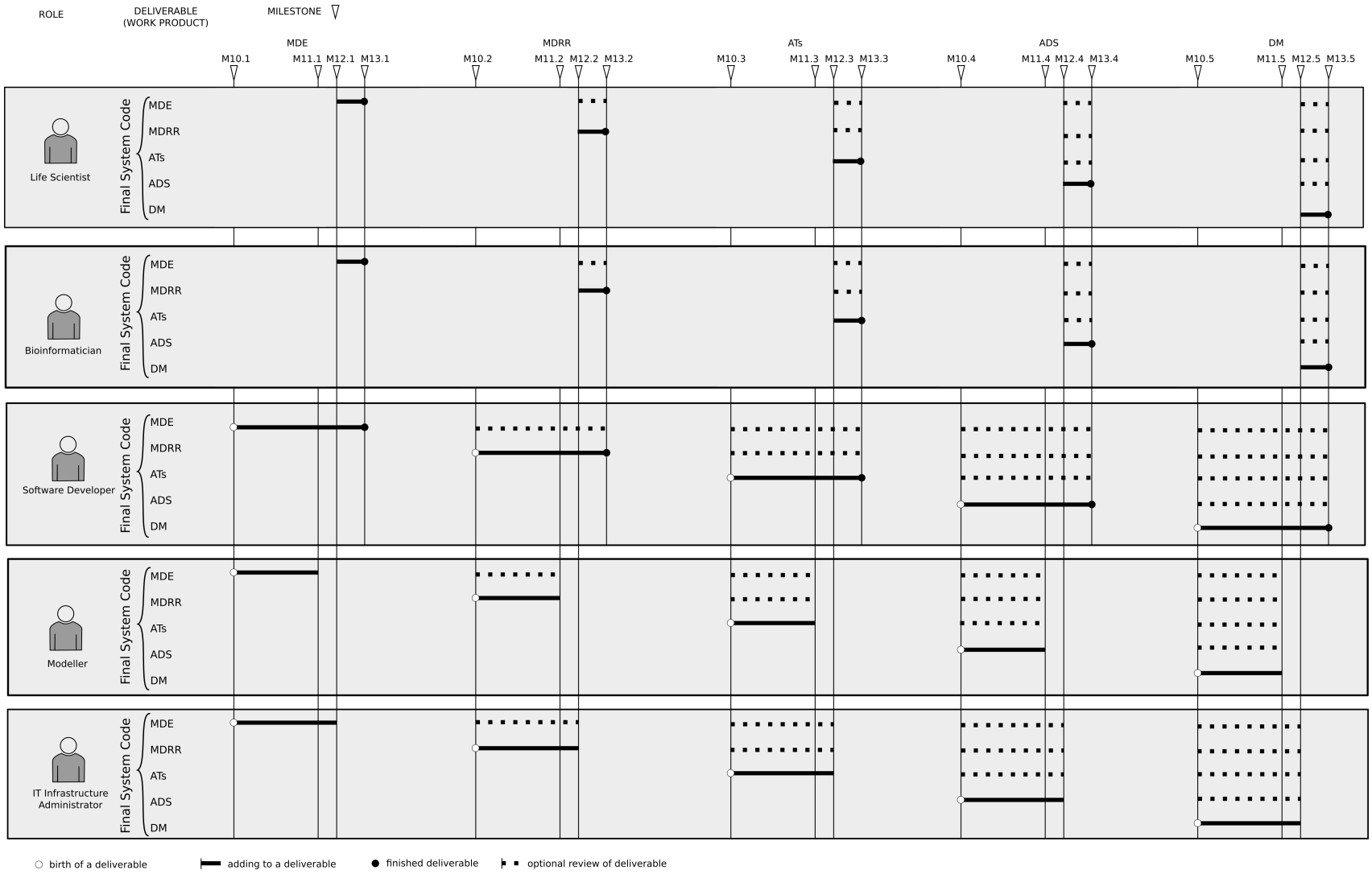

Figure 7.2: Role-Deliverable-Milestone diagram (low level)

7.2.2 Low-level diagram

The low-level Role-Deliverable-Milestone diagram (Figure 7.2) describes the development of the final system code. For instance, even thought the development of the MDE should be the first element to be developed, during the development of all the following elements, the MDE can be altered at specific times as the dotted lines show. A similar approach is used for developing elements that belong to the other element types. During the revision of elements, any missing interfaces can be implemented.

This approach assumes the elements belonging to different element types can be implemented individually and then combined into a working system.

7.3 Methodology techniques

Using the framework assumes additional design and planning activities, which are the main focus of the high-level Role-Deliverable-Milestone pictorial view. They include creating abstract descriptions of the entire scenario (using the basic diagram of the framework), as well as a detailed diagrammatic, textual, and formal description of the elements and their interfaces. In summary, following the methodology which uses the framework provides these three techniques:

1. The basic diagram of the entire scenario can be used to determine whether the scenario fits the framework.

2. The abstract description of an element can be used to determine whether there is a similar already implemented element that can be reused.

3. The abstract description of a prospective tool can be used to determine whether it can be used as an element in a scenario.

7.4 Using the methodology for the implementations

The different roles specified in the methodology represent an ideal situation. In the implementations of the three scenarios, the candidate took the roles of Bioinformatician, Modeller, and Software Developer; the role of Life Scientist was taken by Hans Heindl and Pamela Greenwell (PhD Candidate and Principal Lecturer in Biomedical Sciences at UoW, respectively); and the role of IT Infrastructure Administrator was taken by Juha Hemminki and Hannu Visti, cloud administrators at UoW.

The abstract descriptions (milestones M1 - M9) were created after several meetings of the team. The abstract descriptions of elements draw inspiration from one another and can be derived from the abstract descriptions of the appropriate element type. All three scenarios use the cloud-enabled version of a docking tool (Chapter 3) as MDE. Its implementation (milestones M10.1 - M12.1) included input from all above-mentioned roles.

In an internal presentation, the Life Scientists provided feedback and suggestions for improvement in the way the tool reported on ongoing computation (equivalent to milestones M12.1 - M13.1). After a second iteration of coding (M10.1 - M12.1) the Life Scientists were involved in another feedback session and suggested the inclusion of default input files to improve usability (M12.1 - M13.1). This was implemented in a third iteration representing milestones M10.1 - M12.1. The three scenarios used a variation of the same custom-made MDRR, as it will be elaborated in the next chapter. The definition of the database structure included multiple inputs from Life Scientists, representing milestones M12.2 - M13.2. A similar approach was used to create the specific ATs and the DM.

7.5 Conclusion

This chapter has introduced a light-weight agile methodology which contains the diagrammatic, textual, and formal description of elements and interfaces, as specified in the framework. In line with the guidelines for creating small and light methodologies, shown in [233], three examples of describing a scenario using the recommended abstract descriptions of elements and interfaces will be provided in this thesis as examples of work products. This methodology does not have a detailed description of the concepts used, such as the Z notation. In order to write the methodology by role, so that each team member undertaking a role will know what tasks are required from them, a version of the Role-Deliverable-Milestone pictorial view was used. As shown by the low-level and high-level Role-Deliverable-Milestone pictorial views, this methodology includes active involvement of the end-users - the life scientists.

The methodology specifies three techniques which will be justified in Chapter 8 by implementing three scenarios using the framework and methodology. It will focus on showing how the three techniques mentioned above can be used in real-life scenarios.

Chapter 8

Evaluation

8.1 Introduction

The proposed framework (Chapters 4, 5 and 6) and methodology (Chapter 7) will be evaluated in this chapter. In order to show the usefulness of the framework, three of the five scenarios obtained from the interviews are implemented.

Following the methodology means using the three techniques described in Section 7.3. For each of the three scenarios, the problem was split into the defined elements (MDE, MDRR, AT, ADS, and DM), and interfaces between them. The abstract descriptions of each required element were created. Often the formal description was derived after analysing the diagrammatic description of the element.

The abstract description of the entire scenario can be compared to the abstract description of the framework to determine whether it fits the framework. By browsing through a library of abstract descriptions of already implemented elements, same or similar elements from other scenarios can be examined. If an already implemented element fits the new scenario, it can be reused. As the implementations that follow will show, practically the same code of an implemented element can be used in a different scenario. Finally, if one is not certain whether an existing tool can be used in the implementation, its abstract description can be compared to abstract descriptions of element types. This can show if the tool can be used.

The coding of each scenario was guided by the abstract descriptions. An overview of the coding of each scenario will be provided in this chapter. Using the framework and methodology should make the implementation of the scenarios more reliable, less error-prone, and easier to use. The proper use of the abstract descriptions and the methodology techniques (Section 7.3) would make it more reliable and less error-prone. In order to

show that the implementations are easy to use and usable by biomedical scientists, this chapter includes several usability tests of the implementations.

Three of the scenarios defined through the interviews will be implemented in this chapter. There were several reasons for choosing these particular scenarios. Implementing each of the five scenarios would prove several specific aspects from the framework's point of view. The implementation of Scenario 1 proves that the framework and methodology can be used to obtain a useful prototype. It also shows that existing tools can be used as MDE and AT, while custom-made elements of the types MDRR, AT, and DM can be created too. Implementing each of the other scenarios provides additional proofs when compared to Scenario 1, namely:

- Implementing Scenario 2 proves that:
  1. A new element type not used in Scenario 1 (ADS, e.g. PubChem) can be used.
  2. Elements from Scenario 1 (AT:AssessDocking, MDE:Raccoon2 and the MDRR) can be reused.
- Implementing Scenario 3 proves that:
  1. An element not used in Scenario 1 (AT:EstimateActiveSite) can be used.
  2. A new element type not used in Scenario 1 (ADS, e.g. wwPDB) can be used.
  3. Elements from Scenario 1 (MDE:Raccoon2 and the MDRR) can be reused.
- Implementing Scenario 4 proves that:
  1. Elements not used in Scenario 1 (AT:LIGSIFT, AT:AssessLIGSIFT and AT:CompareConfig) can be used.
  2. Elements from Scenario 1 (AT:DeepAlign, AT:AssessDeepAlign, MDE: Raccoon2, and the MDRR) can be reused.
- Implementing Scenario 5 proves that:
  1. An element not used in Scenario 1 (AT:CompareDockingResults) can be used.
  2. Elements from Scenario 1 (MDE:Raccoon2 and the MDRR) can be reused.

The three scenarios that will prove most aspects were chosen: Scenario 1, Scenario 2, and Scenario 4. Implementing Scenarios 3 and 5 would not add anything that will not be proven by implementing these three chosen scenarios.

These three scenarios will illustrate the use of the three methodology techniques (Section 7.3) as shown in Figure 8.1. Since there are no elements that have been already implemented using the framework, the implementation of Scenario 1 can only utilise Technique 3. On the other hand, when implementing Scenario 2, Technique 2 enables the reuse of

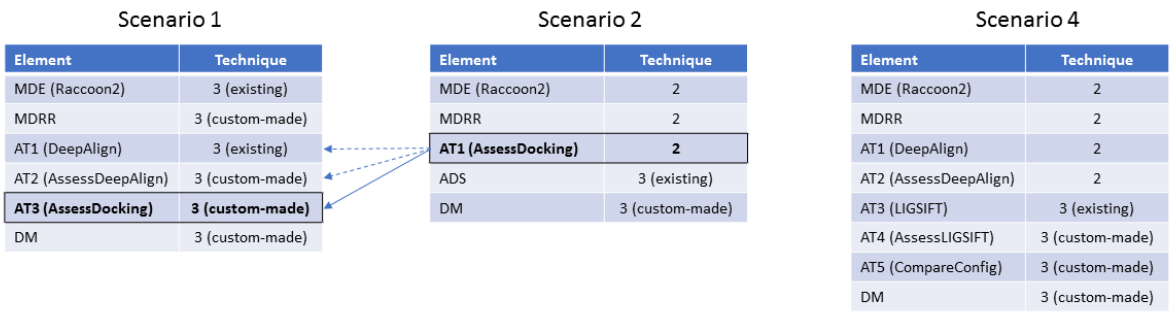

Figure 8.1: Overview of techniques used in the three selected scenarios.

the already implemented MDE and MDRR. The most interesting use of the framework can be observed when implementing the AT for assessing docking results which is required in Scenario 2. At this point, there are three already implemented ATs that are candidates for reuse. The AT: AssessDocking, which is AT3 in Scenario 1, can be reused since it has the same core computation and the same types of interfaces as the required AT in Scenario 2 (the dashed arrows in Figure 8.1 show the candidates which were considered but not chosen, and the full arrow shows the chosen AT). An analogous method can be used when implementing Scenario 4. The details of these procedures will be described in the remainder of this chapter.

8.2 Implementing Scenario 1

The title of Scenario 1 is: Suggest a ligand-protein pair that should be used in the next molecular docking, based on protein similarity and previous results. Scenario 1 starts with the user running a docking or VS simulation. Previous docking results should then be analysed to find receptors that are similar to the currently used receptor. Once a similar receptor has been found, the previous docking results of that receptor should be analysed to filter out ligands which have been successfully docked to it. This ligand (or multiple ligands) can be suggested as a candidate for the next docking with the currently used receptor. A basic diagram of Scenario 1 was shown in Figure 4.3. It proposes the use of six elements of these four element types:

- MDE: The extension of Raccoon2 presented in this thesis.
- MDRR: A custom-made MongoDB-based repository.
- 3 × AT: the structural alignment tool DeepAlign (AT1), a custom-made assessor of DeepAlign (AT2), and a custom-made assessor of docking results (AT3).
- DM: a custom-made DM.

This diagram was derived from the basic diagram of the framework, thus it is reasonable to assume that Scenario 1 fits the framework. However, to provide a more precise analysis, an attempt to derive a detailed diagram for each element and its interfaces can be made. Prior to starting the coding step, a textual and formal description of each element and its interfaces can be used to confirm that the proposed elements can be used.

8.2.1 Abstract descriptions of Scenario 1

MDE: The extended version of Raccoon2 Any existing docking tool can be used as an MDE, as long as it fits the description of the element type MDE of the framework. The extended version of Raccoon2, as described in Chapter 3, is one option. In order to to determine whether Raccoon2 can be the MDE, a diagrammatic, textual, and formal description of the interfaces of Raccoon2 as an MDE can be created.

If a diagram of Raccoon2 can be derived from the detailed diagram of the element type MDE (Figure 6.2), if the list of interfaces of Raccoon2 can be derived from the list of interfaces of an MDE, and if the formal description of Raccoon2 and its interfaces can be derived from the formal description of an MDE and its interfaces (Appendix B), then one can conclude that Raccoon2 can be used as an MDE.

Figure 8.2 shows the detailed diagram of the extended version of Raccoon2 with gUSE. This diagram shows that replacing the generic labels in Figure 6.2 with Raccoon2-specific ones is possible. The required user input for docking consists of one or more ligands, receptors and appropriate configuration files. The result of the docking can be provided to the user, and forwarded to an element representing an MDRR. For Scenario 1, two additional user input values are required: AutoDock Vina affinity threshold and DeepScore threshold. These can be entered into Raccoon2 and forwarded to the MDRR. The suggested candidate ligand for next docking, provided as a result by the MDRR, can be presented to the user. A list of interfaces, derived from the list of MDE interfaces, can also be created.

Raccoon2 interfaces

1a-c. Raccoon2 provides a user interface to obtain ligand, receptor, and config files.

1d-e. Raccoon2 should provide a user interface to obtain the AutoDock Vina and DeepScore thresholds.

2. Raccoon2 provides a user interface to view docking results which should be extended to include the suggested ligand for next docking.

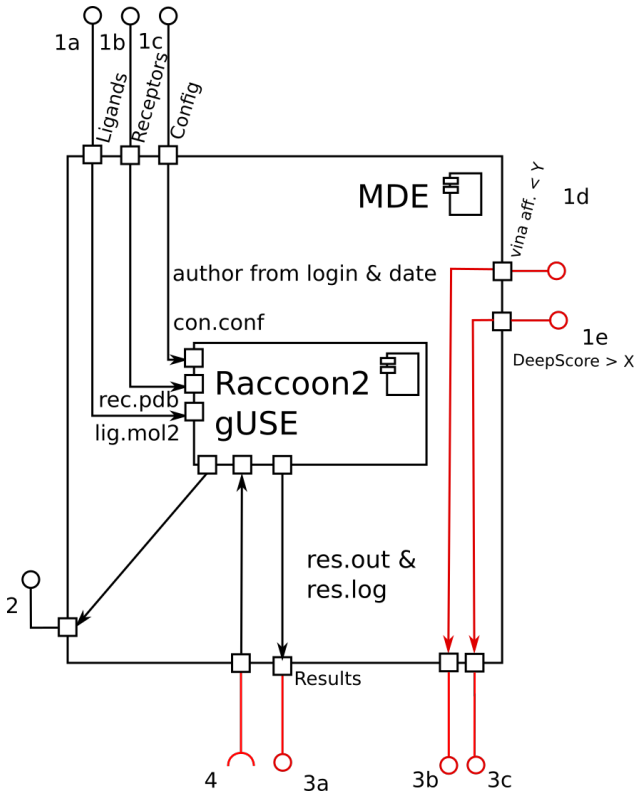

Figure 8.2: The diagram of the MDE: the cloud-enabled Raccoon2 (in red: segments that need to be implemented, in black: existing segments).

3a-c. Raccoon2 needs to provide an interface to the docking results, as well as AutoDock Vina and DeepScore thresholds which should be sent to the MDRR.

4. Raccoon2 requires an interface to receive the suggested ligand for next docking from the MDRR.

Formal description of Raccoon2 The formal description of Raccoon2, shown in Appendix C, was derived from the formal description of the element type MDE (Figure 6.7). The schema *Docking_AutoDockVina* uses *dockingWithConfig*, and has been included in the schema *MolecularDockingEnvironment_Raccoon2* (pages 172 - 173). This is equivalent to the *Docking* and *MolecularDockingEnvironment* schemas from the framework (page 163).

In the generic formal description the element type MDE uses the schema *Docking* to show that a docking can be conducted either without a configuration file (using *dockingWithoutConfig*) or with a configuration file (using *dockingWithConfig*). When describing the specific element Raccoon2 (Figure 8.3), only the definition for *dockingWithConfig* was used, because docking with Raccoon2 uses AutoDock Vina which requires a configuration file. The schema *Docking_AutoDockVina* is derived from *Docking* by leaving out the option to use *dockingWithoutConfig* and changing the schema's name. The schemas *MolecularDockingEnvironment_Raccoon2* and *ViewMolecularDockingEnvi-*

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking_AutoDockVina

$$ligand? : LIGAND$$
$$receptor? : RECEPTOR$$
$$config? : CONFIG$$
$$result! : RESULT$$
$$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$$

MolecularDockingEnvironment_Raccoon2

$$ligands? : LIGANDS$$
$$receptors? : RECEPTORS$$
$$config? : CONFIG$$
$$results! : RESULTS$$
$$date! : DATE$$
$$\exists ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking\_AutoDockVina$$

ViewMolecularDockingResults_Raccoon2

$$\Xi MolecularDockingEnvironment\_Raccoon2$$
$$results! \neq \varnothing$$

Figure 8.3: Excerpt of the Z notation describing Raccoon2 as element of Scenario 1.

ronmentResults_Raccoon2 are the same as the respective generic schemas in all but name.

MDRR: a custom-made MongoDB repository An existing repository can be used as an MDRR, as long as it fits the description of the MDRR element type of the framework. To the best of the candidate's knowledge, no such repository exists. Furthermore, since this is the first scenario implemented using the framework, there is no library of abstract descriptions of existing tools to use for comparison. Creating a custom-made repository that would be used as an MDRR is one solution. The abstract descriptions of the proposed custom-made tool is shown, while Sub-section 8.2.2.2 shows the benefits of using MongoDB as a database.

Figure 8.4 shows the detailed diagram of the proposed custom-made tool by replacing the generic labels of Figure 6.2 with specific ones. The MongoDB-based MDRR would require docking results. It would also require the AutoDock Vina and DeepScore threshold values,

which should be forwarded to an element that uses them. Any other data stored in the repository should also be provided for the next elements. The MongoDB-based MDRR requires the input of the ATs in order to keep track of the process, and the DM in order to store the suggested ligand for next docking. This suggestion should be provided to the MDE and subsequently viewed by the user. A comprehensive list of interfaces, derived from the list of MDRR interfaces, can also be created.

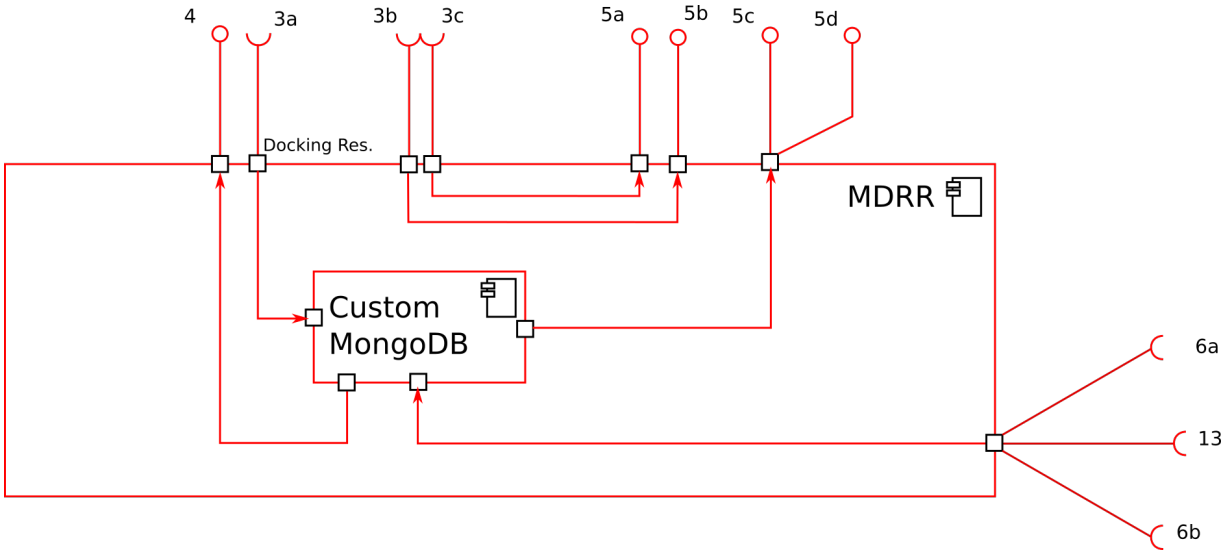

Figure 8.4: The diagram of the custom-made MongoDB-based MDRR.

Interfaces of the custom-made MongoDB-based MDRR

3a-c. The MDRR should require docking results, AutoDock Vina threshold and DeepScore threshold.

4. The MDRR needs to provide the suggested ligand for next docking to the MDE.

5a-b. The MDRR needs to provide the DeepScore threshold to the DeepAlign AT, and the AutoDock Vina threshold to the docking assessment AT.

5c. The MDRR needs to provide a list of all receptors stored in the repository, and the currently used receptor, to the DeepAlign AT.

5d. The MDRR needs to provide a list of previous docking results that have used receptors similar to the current receptor, to the docking assessment AT.

6a-b. The MDRR should require the results from the DeepAlign AT and the docking assessment AT to keep track of the process.

13. The MDRR needs to receive the suggested ligand for next docking from the DM.

Formal description of the custom-made MongoDB-based MDRR The formal description of the proposed custom-made MongoDB-based MDRR (Appendix C, pages 173 - 175), is virtually the same as the description of the element type MDRR. The schema *MolecularDockingResultsRepository_MongoDB* is a replica of *MolecularDockingResults-Repository* from the framework's description, albeit with an altered name. It shows that the repository should store data about the ligand, receptor, configuration file, date and docking result. The fact that the suggestion of ligand for next docking is also stored in the MDRR is modelled using the *decisionRepository*. The data can be inserted into or selected from these model repositories.

AT1: DeepAlign Any existing structural alignment tool can be used as an AT in Scenario 1, as long as it fits the description of the element type AT. Chapter 2 provided an overview of several existing tools, and proposed using the tool DeepAlign. An abstract description of DeepAlign can be created to determine whether it can be the AT. A diagrammatic, textual, and formal description of the interfaces of DeepAlign as an AT will be created. Similarly to the way it was concluded that Raccoon2 can be the MDE, if the diagram, list of interfaces, and formal description of DeepAlign can be derived from the abstract descriptions of the element type AT (Figure 6.4, and Appendix B), then one can conclude that DeepAlign can be used as an AT.

Figure 8.5 shows the detailed diagram of DeepAlign. It shows that replacing the generic labels in Figure 6.4 with DeepAlign-specific ones is possible. A user-provided threshold of the value of DeepScore is required, along with a list of all previous receptors and the currently used receptors which will be compared. The threshold and the results of the pairwise comparison should be sent to another AT which will assess whether the structural similarity score is sufficient to call two receptors similar. A more comprehensive list of interfaces, derived from the list of AT interfaces, can also be created.

Interfaces of DeepAlign

5a. The DeepAlign AT should require the DeepScore threshold.

5c. The DeepAlign AT should require a list of receptors, and a target receptor to calculate the structural alignment.

7a. The DeepAlign AT should provide the DeepScore threshold to an assessment AT.

7b. The DeepAlign AT should provide the structural alignment results along with any meta-data for assessment.

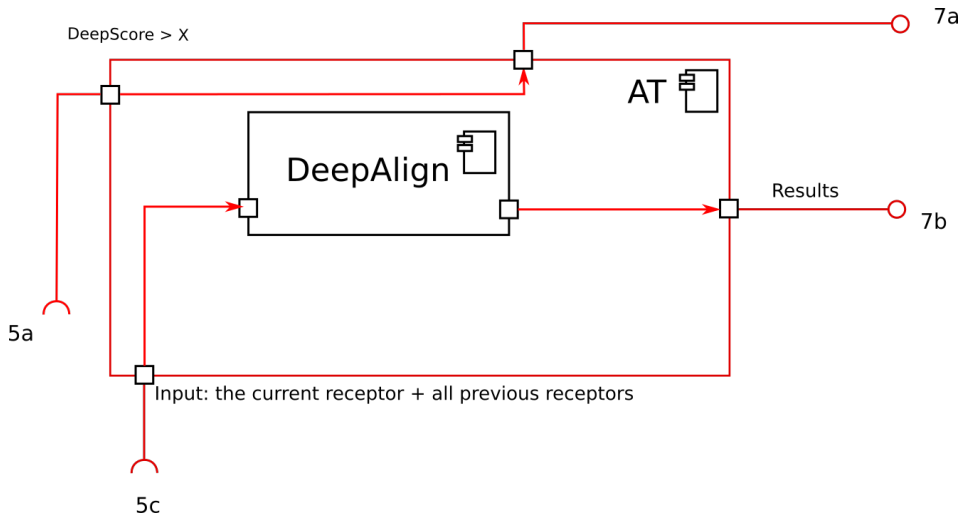

Figure 8.5: The diagram of the AT DeepAlign.

Formal description of DeepAlign The generic formal description of an AT included all possible classes of ATs based on the input they receive. The schema *DeepAlign* (Appendix C, page 176) is derived from the schema *AdditionalTool* of the framework's formal description (Appendix B, page 168). The schema *AdditionalTool* showed that the result of the AT can come from any combination of AT classes based on the input. The schema *DeepAlign* specifies that this needs to be *DeepAlignCore*, a tool that uses previous results, equivalent to *additionalTool_PR* of the framework. Instead of generic previous results, it specifically defines the input as a pair of receptors. In *DeepAlignCore*, it is shown that a *current_receptor* and a *previous_receptor* need to exist, in order for a DeepAlign result based on these two receptors to exist.

AT2: Assess DeepAlign results In an analogous manner to the above elements, it has been decided to use a custom-made tool to assess the results of DeepAlign and filter receptors that are "similar" to the currently used receptor. Since there is no library of already implemented elements, it is reasonable to propose creating a custom-made tool that will do the assessment based on a user-provided threshold of the DeepScore value. Abstract descriptions of this custom-made tool can be derived from the abstract descriptions of the element type AT. The detailed diagram is shown in Figure 8.6. The list of interfaces is as follows.

Interfaces of the custom-made threshold-based DeepAlign assessment AT

7a-b. This AT should require the DeepScore threshold and DeepAlign result (along with any meta-data) as input.

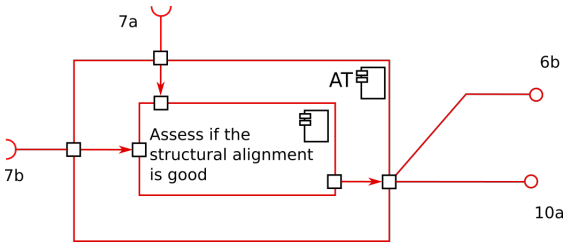

Figure 8.6: The diagram of the AT to assess DeepAlign.

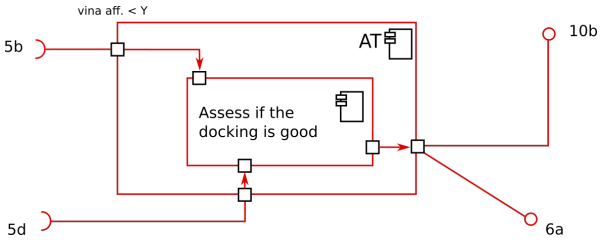

Figure 8.7: The diagram of the AT to assess docking results.

6b. This AT should provide the results of the assessment (whether the DeepAlign result is sufficient to label the two receptors as similar) to the MDRR for storage.

10a. This AT should provide the results of the assessment to the DM for summarising.

Formal description of the custom-made threshold-based DeepAlign assessment AT The schema *AssessDeepAlign* (Appendix C, page 176) is derived from the schema *AdditionalTool* of the framework's formal description (Appendix B, page 168), and *goodDeepAlignResult* is equivalent to *additionalTool_UI_ATR* which uses a user-provided input and results of another AT. In *goodDeepAlignResult* it is explained that the user-provided input needs to be a threshold which will be compared to the DeepScore value of the DeepAlign result. The schema *AssessDeepAlign* shows that a receptor r will be part of the receptors assessed as similar only if the result of *goodDeepAlignResult* is positive.

AT3: Assess docking results An analogous method was used to propose a custom-made tool to assess the docking results and filter "good" docking results. This is useful because the scenario requires a similar receptor which has been "successfully" docked with a ligand. In Scenario 1, there is no library of already implemented elements, so a custom-made tool could conduct the assessment based on a user-provided threshold of the AutoDock Vina affinity value. Figure 8.7 shows the detailed diagram which, along with the other abstract descriptions of this custom-made tool, can be derived from the abstract descriptions of the element type AT. The list of interfaces is as follows.

Interfaces of the custom-made threshold-based docking result assessment AT

5b, 5d. This AT should require the AutoDock Vina threshold and a list of docking results.

6b. This AT should provide the results of the assessment (whether the docking result is good) to the MDRR for storage.

10b. This AT should provide the results of the assessment to the DM for summarising.

Formal description of the custom-made threshold-based docking result assessment AT The schema *AssessPreviousDocking* (Appendix C, page 177) is derived from the schema *AdditionalTool* of the framework's formal description (Appendix B, page 168), and *goodDocking* is equivalent to *additionalTool_UI_PR* which uses a user-provided input and previous results. In *goodDocking*, it is shown that the user-provided input needs to be a threshold which will be compared to the docking score of the previous docking result. The schema *AssessPreviousDocking* shows that a previous docking *result* will be part of the docking results assessed as similar only if the result of *goodDocking* is positive.

DM: custom-made element The DM combines the result of AT2 (if receptors are similar) and AT3 (if the docking is good). It should create a list of receptors sorted by alignment score first (most similar to the current receptor), then by the AutoDock Vina score (part of best docking results). Based on this sorted list of receptors, the MDRR can select ligands that have been docked with them and suggest these ligands for a next docking. The DM is an element type that is very specific to every scenario, so it will likely be a custom-made tool. Nevertheless, abstract descriptions of this custom-made tool can be derived from the abstract descriptions of the DM element type. The detailed diagram is shown in Figure 8.8. The list of interfaces is as follows.

Interfaces of the custom-made DM

10a-b. The DM should require the result from AT2 and AT3.

13. The decision, in this case the sorted list of most similar receptors that were part of the best docking results, should be provided to the MDRR.

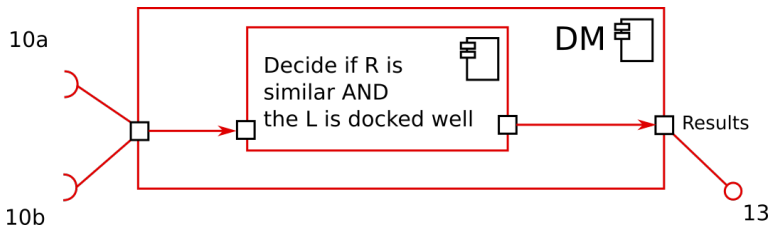

Figure 8.8: The diagram of the DM.

Formal description of the custom-made DM The formal description of the custom-made DM is derived from the formal description of the DM element type of the framework. The schema *DecisionMaker_Custom* (Appendix C, page 177) is equivalent to the schema *DecisionMaker* of the generic element type DM (Appendix B, page 171). It specifically uses the *makeADecisionAdditionalTool* which requires the results of two ATs as input. In

this case the first result is a list of filtered receptors and the second is a list of filtered previous docking results.

Detailed diagram of entire Scenario 1 Based on the analysis shown above, a complete detailed diagram of Scenario 1 can be created (Figure 8.9). It confirms that Scenario 1 fits the framework because it is equivalent to the detailed diagram of the entire framework (Figure 6.1). The items drawn in red are ones that need to be implemented, while the ones in black are existing items that can be used. There are a total of 21 interfaces between these elements (numbered according to the numbering-scheme of the framework).

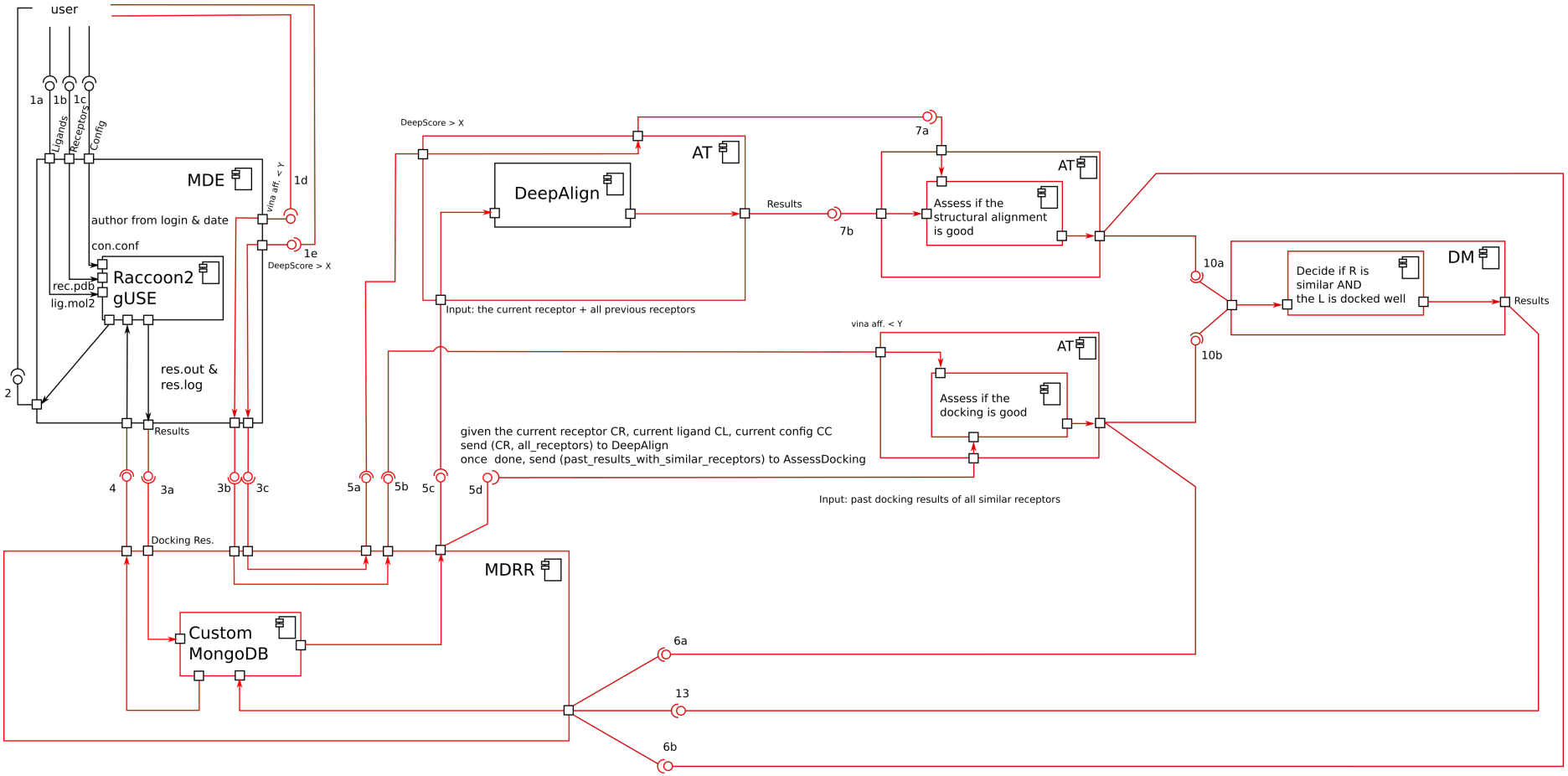

Figure 8.9: The detailed diagram of Scenario 1.

8.2.2 Code of Scenario 1

Once the diagrammatic, textual, and formal description of the elements and interfaces of Scenario 1 have been created, the coding step can begin. There were several software engineering decisions made for this implementation. The framework and methodology are independent of these decisions. For instance, all the elements in this implementation can be accessed using a simple RESTful API through HTTP. To implement this API, the minimalist web framework "Bottle" [236] was used. Bottle is a Python framework that enables an easy setup of a server. Python was chosen as the programming language merely to provide continuity, because the Raccoon2 software has been developed in Python. Bottle is very simple to use and convenient for prototyping solutions, hence it was chosen to build this prototype implementation. To decrease communication delays between elements, elements are grouped in 3 servers. This implementation of Scenario 1 first filters out similar receptors, then it searches for good docking result of those receptors.

Figure 8.10 provides more details about the flow and communication between the elements and the servers. In order to insert the results of the current docking or VS simulation, the MDRR on Server 1 expects the results of Raccoon2 as POST parameters. It processes the results (using the Parser) and inserts information from them into the MongoDB database, which includes the collections *results*, *receptors*, *ligands*, and *analysis*. The functions for inserting data in the database are grouped in the Inserter, while the functions for selecting data are in the Selector.

Once done with inserting, the MDRR returns a response to the MDE (Raccoon2). Another HTTP request is sent to obtain a suggestion of ligands for the next docking. The MDRR selects "all receptors" from the database and sends them to the AT:DeepAlign on Server 2, along with the current ("target") receptor, and a "threshold" value of DeepScore which is input by the user within Raccoon2.

The AT:DeepAlign on Server 2 executes DeepAlign for each pair of receptors. It then calls the AT:AssessDeepAlign, located on the same server, in order to filter out "similar" receptors. This AT assesses the DeepAlign results by comparing the value of DeepScore to the user input threshold. If the DeepAlign result is greater than the threshold, the two receptors are called "similar". A list of these "similar" receptors is returned to Server 1 and it is then used by the DM. Once the "similar" receptors have been received, Server 1 inserts several documents to the "analysis" collection to keep track of all events. Then, only previous docking results where the receptor is one of the similar receptors are selected, and sent to the AT:AssessDocking on Server 3, along with a second threshold value of the AutoDock Vina "affinity".

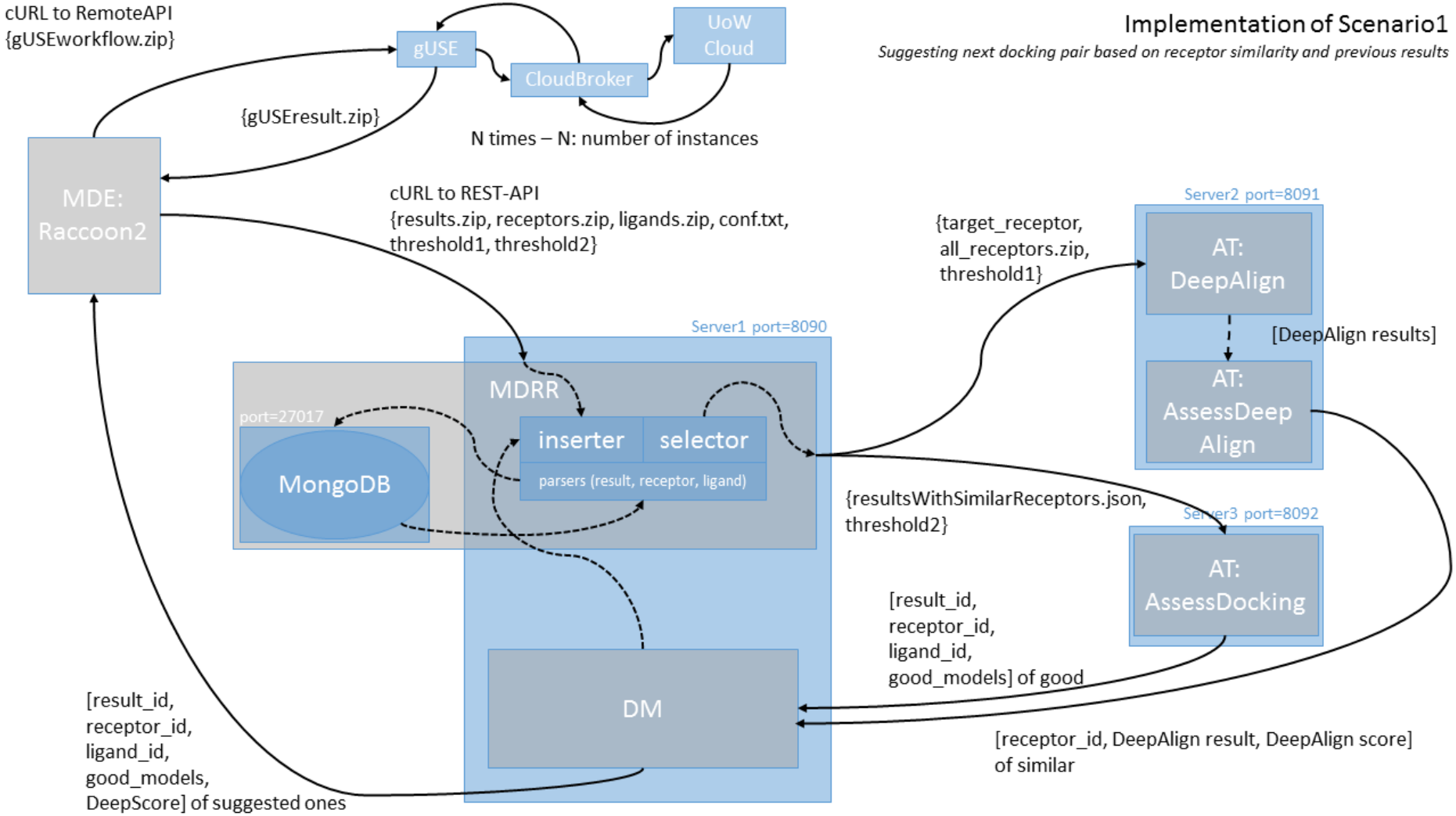

Figure 8.10: Communication between servers used in the implementation of Scenario 1.

The AT:AssessDocking on Server 3 searches through the docking results for a result that has at least one docking model where the AutoDock Vina affinity is less than the threshold, and calls this a "good" docking result (an AutoDock Vina docking result usually contains 10 models). When the assessment is completed, Server 1 receives a response and inserts a document in the "analysis" collection. It then initialises the DM on the same server and sends it the "similar receptors" and the "good results".

The DM on Server 1 combines these two lists, and sorts the list of results firstly based on the DeepScore value of its receptor, and then on the affinity value of the docking results. An ordered list of suggestions is formed with the first element being a ligand which has the best docking result with a receptor that is the most similar to the currently used one.

More details, along with the entire source code, is provided on GitHub [237]. The remainder of this section will focus on how the abstract descriptions of elements and interfaces are reflected in the code.

8.2.2.1 Implementation of the MDE

The extended version of Raccoon2 (Chapter 3) can be used as an MDE in Scenario 1. As shown by the detailed diagram (Figure 8.2), Raccoon2 needs to be further extended to ask the user for two additional values: the DeepScore, and the AutoDock Vina threshold.

8.2.2.2 Implementation of the MDRR

The MDRR for Scenario 1 can be a custom-made MongoDB-based [201] repository. There were several reasons for choosing MongoDB as the underlying database engine:

1. The schema-less design is ideal for polymorphic data.
   (a) Structures of ligands and receptors can be stored in a collection regardless of the file format (be it .mol2, .pdb, or something else).
   (b) Docking results can be stored in a single collection regardless of the docking tool and file format of the result files.
   (c) One collection can be used for keeping track of all activities (provenance information) regardless of the type of activity, AT or decision made.
2. MongoDB scales very well for large amounts of data, provided it is well designed and features such as sharding and indexing are utilised.
3. It is well-suited for prototyping because it is easier to change what is stored during development.

Polymorphic data is data that changes structure. Because different scenarios could require a different DM, AT, or MDE (therefore different docking tool), and different format of docking results, this type of software systems are ideal for MongoDB. Furthermore, if an element is swapped for another element that has a different format of results, the same collection (a MongoDB "collection" is somewhat equivalent to an SQL table) can be used. Python is a good choice because there is a MongoDB driver (PyMongo [238]), and two well-maintained Python libraries for calculating molecular properties: openbabel and pybel [239].

The structure of the database has been derived from the formal description and the detailed diagram of Scenario 1. After analysing the schema *MolecularDockingResultsRepository_MongoDB* (Appendix C, page 173), it becomes clear that the MongoDB database should contain data about ligands, receptors, docking results, the date, and the decision (in this case the suggested ligands for the next docking). Furthermore, the schema *SelectMolecularDockingResults* shows that the database should provide different ways to select data. The number and type of collections in the MongoDB database have been mainly derived from these two Z schemas. Due to the inherent difference in the type of data stored, separate collections for *ligands*, *receptors*, and docking *results* have been created. The date when the docking was conducted can be uploaded along with the docking results. Therefore, the date is a property in the collection called results. A fourth collection, called *analysis*, has been created to store the decision.

The diagram of the entire Scenario 1 shows that the MDRR requires 4 interfaces, thus 4 types of input: the docking results, the decision from the DM, the results of AT:AssessDeepAlign, and the results of AT:AssessDocking. This diagram made it evident that the *analysis* collection should also store the results of the ATs. Because the results of the ATs, and the result of the DM (the decision) are often going to be different based on the scenario, it is impractical to create a new collection for each type of AT or DM. Having a single collection means that if one AT element is changed for another, there will not be a need to change the structure of the database. If another structural alignment tool is used instead of DeepAlign, the format of the result would be different, but it could still be stored in the collection called analysis. Finally, this means that the same database structure can be used for another scenario which would have different ATs.

Details of the MongoDB collections used are not provided here, but can be obtained from the source code [237]. The results collection contains the date, and all relevant information from an AutoDock Vina docking result file. If another docking tool is used instead of AutoDock Vina, the format of the docking results would change, but the same *results* collection can remain in use.

8.2.2.3 Implementation of the ATs

AT:DeepAlign and AT:AssessDeepAlign To improve the efficiency of this implementation, these two ATs have been implemented on the same server, as separate modules. The detailed diagram of Scenario 1 shows that the results of the AT:DeepAlign should be sent to the AT:AssessDeepAlign. The formal description of Scenario 1 confirms this by showing that the Z schema *AssessDeepAlign* requires the method *goodDeepAlignResult* which uses *DEEP_ALIGN_RESULT* as input. These two abstract descriptions of the ATs have been used to write the scripts that represent the ATs.

Since they reside on the same server, the same Bottle "controller" (controller.py) is used to launch a new thread representing the AT:DeepAlign for each receptor pair. DeepAlign is run and the results are sent to an object of another class representing AT:AssessDeepAlign. The value of DeepScore is extracted and compared to the user-provided threshold.

AT:AssessDocking The methodology specified that before implementing a custom-made tool, one can search a library of abstract descriptions of previously implemented elements for an element that can be reused. This is the first time that an AT where the core computation is meant to assess docking results, so the already implemented elements AT:DeepAlign and AT:AssessDeepAlign cannot be reused. After comparing the abstract description of the required AT and the generic element type AT, it was concluded that the required AT can be derived from the generic element type according to Technique 3 in Section 7.3. There are some similarities between the required AT and AT:AssessDeepAlign since the method *goodDocking* of the formal description of AT:AssessDockingResults, and *goodDeepAlignResult* of AT:AssessDeepAlign resemble each other. They both require a user-provided threshold which they use to filter out results from a previous tool. In the case of the required AT, the previous tool is a docking tool, and when the threshold is the AutoDock Vina affinity, the more negative the value is the "better" the docking. Therefore, a clear difference is that this AT should filter out docking results with docking score that is "$\leq$" the threshold.

Because of these similarities, the code of the custom-made AT:AssessDocking, which is implemented on a separate server, resembles segments of the code of AT:AssessDeepAlign with some important changes. The specific differences can be seen in the source code [237].

8.2.2.4 Implementation of the DM

The formal description of the DM (Appendix C, page 177) shows that the DM should require results from ATs in the form of a list of receptors and a list of docking re-

sults. The structure of the method `dm.decider.SimpleDecide` has been derived from this segment of the formal description. It expects the `assessed_similar_receptors` and `assessed_results` as input and combines them in a single list where each list item contains data about the similar receptor and its docking results. This list is then sorted firstly by the DeepScore value, then by the AutoDock Vina affinity. Note that this is the case, even though the commands seem to be in the opposite order because of the way the Python function `operator.itemgetter()` works with the parameter `reverse = True`.

The result of the DM is returned to the MDE, and presented to the user. On the left-hand side there is a tree-like dictionary shown, where each element can be expanded. The right-hand side will be populated upon clicking an item (Figure 8.11).

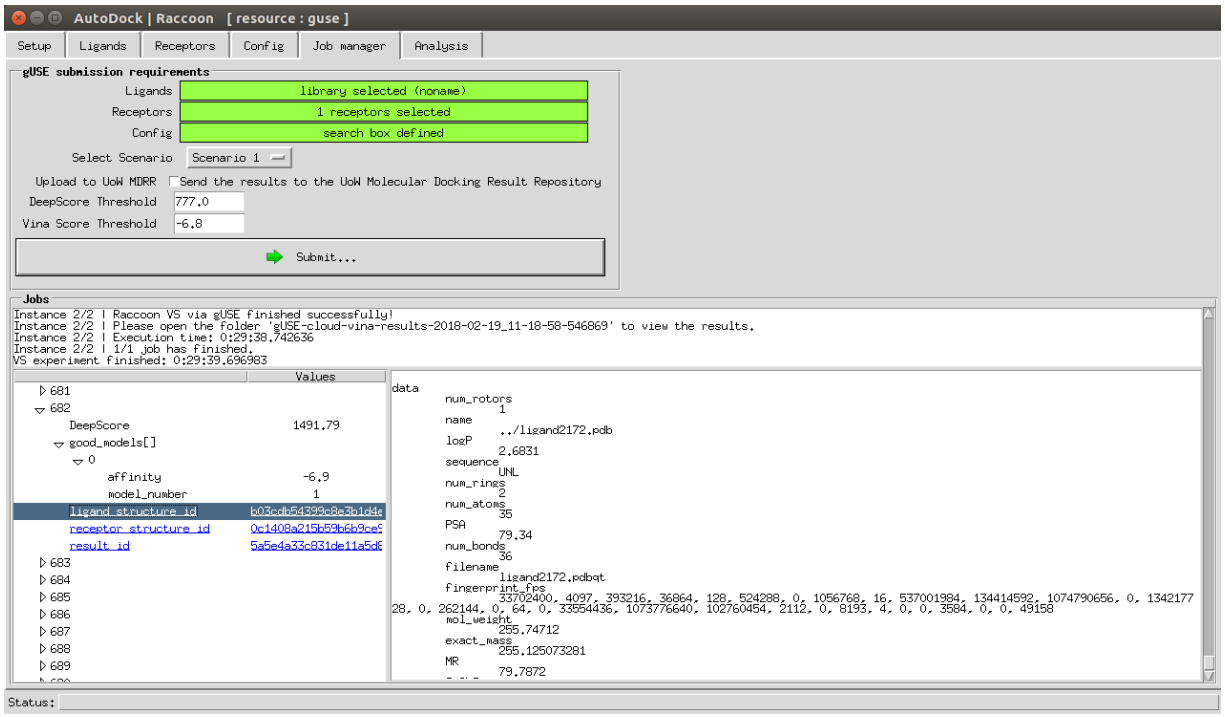

Figure 8.11: Screenshot of the final result of Scenario 1.

8.3 Implementing Scenario 2

The title of Scenario 2 is "Filter suitable results for laboratory experiments, based on ligand properties". Scenario 2 starts with the user running a VS between one receptor and a large number of ligands. Then, ligands of "good" docking results are filtered based on a property that is available in an external additional data source. The final result is a sublist of ligands that are more likely to produce useful laboratory results. A basic diagram of Scenario 2 was shown in Figure 4.4. It proposes five elements of these five element types:

- MDE: The extension of Raccoon2 presented in this thesis.

- MDRR: A custom-made MongoDB-based repository.
- AT: AssessDocking, a custom-made tool that filters' "good" docking results based on a threshold.
- ADS: PubChem, the existing external database of ligand properties.
- DM: a custom-made DM.

Because this diagram was derived from the basic diagram of the framework, it is reasonable to assume that Scenario 2 fits the framework. To provide a more precise analysis, the detailed diagram for each element and its interfaces will be described in this section. A list of the interfaces, and a formal description will be used to confirm that the proposed elements can be used as element types prior to starting the coding step. This section will focus on segments that are different from the description of Scenario 1, and comment on the added value of implementing Scenario 2 using the methodology.

8.3.1 Abstract descriptions of Scenario 2

MDE: The extended version of Raccoon2 The MDE in Scenario 2 can be the same as the one in Scenario 1 (Section 8.2.1). Practically the entire element can be reused with the difference that instead of a DeepAlign threshold as in Scenario 1, now the user inputs a PubChem property name and threshold. Figure 8.14 shows that a detailed diagram of the Raccoon2 extension for Scenario 2 can be derived from the generic diagram of an MDE. More importantly, it shows how the methodology allows the reuse of previously implemented elements. When drawing the detailed diagram, it becomes evident that most of the interfaces are exactly the same, and the core computation (the docking) is the same as in the detailed diagram of Raccoon2 used for Scenario 1. The conclusion is that the abstract description of Raccoon2 is similar enough for it to be used as an MDE in Scenario 2 as well. The fact that the element can be reused is determined now, before any coding begins. It is determined by searching a library of abstract descriptions of already implemented elements (Technique 2 in Section 7.3). At the moment, this library contains only one MDE, but the same method of drawing the detailed diagram and comparing the interfaces and the core computation, can be used to search a large library.

Raccoon2 interfaces Since the Raccoon2 element can be reused, the list of interfaces is nearly the same. The only difference being the need for a PubChem threshold instead of a DeepScore threshold.

Formal description of Raccoon2 The formal description of the Raccoon2 element of Scenario 2 (Appendix D page 179) can be derived in the usual way from the formal description of the element type MDE. The result is the same formal description as in Scenario 1 (Figure 8.3).

MDRR: a custom-made MongoDB repository Similarly, Scenario 2 can reuse the same MDRR as in Scenario 1 (Section 8.2.1). The same technique for searching a library of already implemented elements can be used to determine whether an element can be reused. The detailed diagram of this MDRR (which can be seen in Figure 8.14), is nearly identical to the detailed diagram of the MDRR in Scenario 1 which is the only MDRR present in the library of implemented elements.

Interfaces of the custom-made MongoDB-based MDRR The interfaces of this MDRR are nearly the same as the MDRR in Scenario 1. The only difference is in interfaces 5a-d which are providing the AutoDock Vina affinity and the PubChem property threshold, the list of current docking results, and a list of ligands of "good" docking results.

Formal description of the custom-made MongoDB-based MDRR When deriving the formal description of this MDRR (Appendix D, page 179), the focus is on the interfaces for inserting and selecting docking results. This results in the same formal description as the one for the MDRR of Scenario 1.

AT1: Assess docking results The same technique for searching a library of already implemented elements can be used when implementing this AT (Technique 2 in Section 7.3). When drawing the detailed diagram for this AT, it can be compared to a library of three already implemented elements (as implemented in Scenario 1).

When compared to the AT:DeepAlign, there is a clear difference in the interfaces. AT:DeepAlign needs to send the results and a user-provided threshold to another AT for assessment. Whereas, the needed AT assesses docking results and sends them to an MDRR for storage, and a DM for summarising. When compared to the AT:AssessDeepAlign, there are more similarities in the interfaces. The difference is that AT:AssessDeepAlign requires input from another AT, whereas the needed AT requires input from an MDRR. There is a big difference in the core computation, the AT:AssessDeepAlign filters structural alignment results, and the needed AT should filter docking results. Finally, when comparing the needed AT with the AT:AssessDocking, it is clear that the interfaces are the same (require input from MDRR, provide results to MDRR and DM), and the core computation is the same (assess docking results).

Therefore, it can be concluded that the AT:AssessDocking can be reused. The same name for the AT can be used and the detailed diagram is nearly the same. The only difference is in the naming of the interfaces 5a and 5c.

This comparison was done manually, examining and comparing the abstract descriptions of the required element to the already implemented elements by hand. Ideally, this comparison would be automated and it would use a database of already implemented abstract descriptions (Section 10.2).

Formal description of the custom-made threshold-based docking result assessment AT The formal description of this AT (Appendix D, page 182) has the same method *goodDocking* and Z schema *AssessPreviousDocking* as in Scenario 1.

ADS: PubChem Implementing Scenario 2 will show how a new element type can be used, since in Scenario 1, there was no ADS (as previously stated in Section 8.1). Since there is no library of previously implemented elements of the type ADS, this scenario can use the third technique of the methodology (Section 7.3) and create an abstract description of an existing tool, then compare it to the generic abstract description of the element type to determine whether it can be used.

PubChem is a repository that contains data about chemical substances. It is split into three databases: Substance, Compound, and BioAssay [17, 240]. As of March 2018, the Compounds database contains more than 94 million items. PubChem can be accessed programmatically through the Power User Gateway (PUG) interface [241]. In Scenario 2, PubChem is proposed as the core computation component of an ADS. The detailed diagram of an ADS of the framework shows that an ADS needs to provide the data through an interface for an MDRR and a DM. When creating the detailed diagram of PubChem, the existence of such an interface was checked. Indeed, PubChem's PUG-REST API is an interface provided by PubChem which can be used to read the data it stores. If the PUG-REST API can be used to obtain the value of a ligand property for a list of ligands, as required by the interfaces 5b and 5d, then PubChem can be used as an ADS. If in the coding of Scenario 2, an HTTP request is sent to the PUG-REST API for each ligand in the list, this will be possible.

Therefore, since the PubChem element, as drawn in the detailed diagram (Figure 8.12), can provide and require the needed interfaces as an ADS, and the core computation segment (the Compounds database) contains data about ligand properties, the PubChem element can be the ADS for Scenario 2. However, the value returned by PubChem would need to be additionally compared with the threshold.

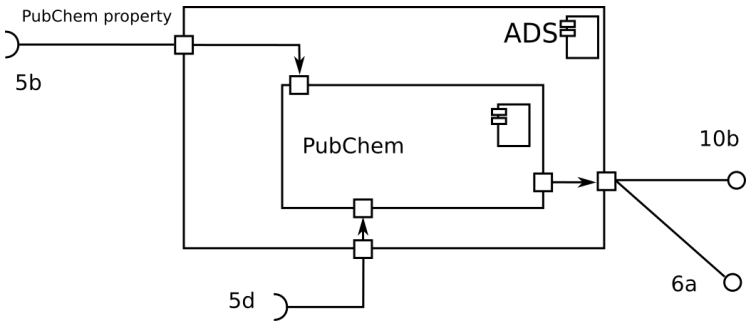

Figure 8.12: The diagram of the ADS PubChem.

PubChem Interfaces

5b, 5d. PubChem requires the user provided PubChem property, and a list of ligands.

6a. PubChem provides the ligand property value, to the MDRR to keep track.

10b. PubChem provides the ligand property value, to the DM for summarising.

Formal description of PubChem This scenario is the first example of a formal description of an ADS element. One of the aims of implementing of Scenario 2 is to show how a new element type, which hasn't been used before, can be introduced (as mentioned in Section 8.1). The abstract description of an ADS in the framework is generic and does not provide details about the type of data a particular ADS element would store. The specific formal description of PubChem (Appendix D, page 182) includes the *checkPubChem* method. This method specifies that it requires a ligand and a property as input. After checking the data stored in PubChem, it provides a positive or negative response based on whether the property of the ligand is within a given threshold. The schema *PubChem*, which is derived from the generic schema *SelectAdditionalDataInfo*, shows how the filtered results based on the ligand property can be obtained.

DM: custom-made element The DM combines the result of AT1 (if the docking is good) and the ADS (if the ligand property is within the threshold). The DM is an element which is specific to each scenario. This can be seen after comparing the detailed diagram of this DM to the DM from Scenario 1. The interfaces seem equivalent, however the main difference is the core computation. Because each scenario would provide a different decision as a final result, the core computation of each DM would be different. In Scenario 2, the DM provides a list of ligands filtered based on a property from an external additional data source. Whereas, in Scenario 1, the DM suggested a ligand for the next docking. Therefore, the DM from Scenario 1 cannot be reused.

Interfaces of the custom-made DM

10a-b. The DM should require the results from the AT1 and the ADS.

13 The decision, in this case the list of filtered ligands according to the specified property, should be provided to the MDRR.

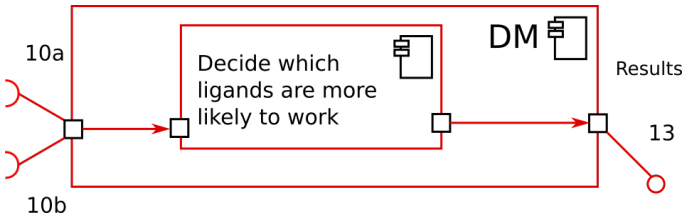

Figure 8.13: The diagram of the DM.

Formal description of the custom-made DM The formal description of this custom-made DM is derived from the element type DM of the framework. The Z schema *DecisionMaker_Custom* (Appendix D, page 183), which is derived from the schema *DecisionMaker* of the framework, specifically uses *makeADecisionPreviousResults*. It requires two previous results as input. In this case the first result is a list of assessed previous docking results, and the second is a list of filtered results based on ligand properties.

Detailed diagram of entire Scenario 2 Similarly to Scenario 1, a complete detailed diagram of Scenario 2 can be created (Figure 8.14). It confirms that Scenario 2 fits the framework because it is equivalent to the detailed diagram of the entire framework (Figure 6.1). There are a total of 19 interfaces between the elements (numbered according to the numbering-scheme of the framework).

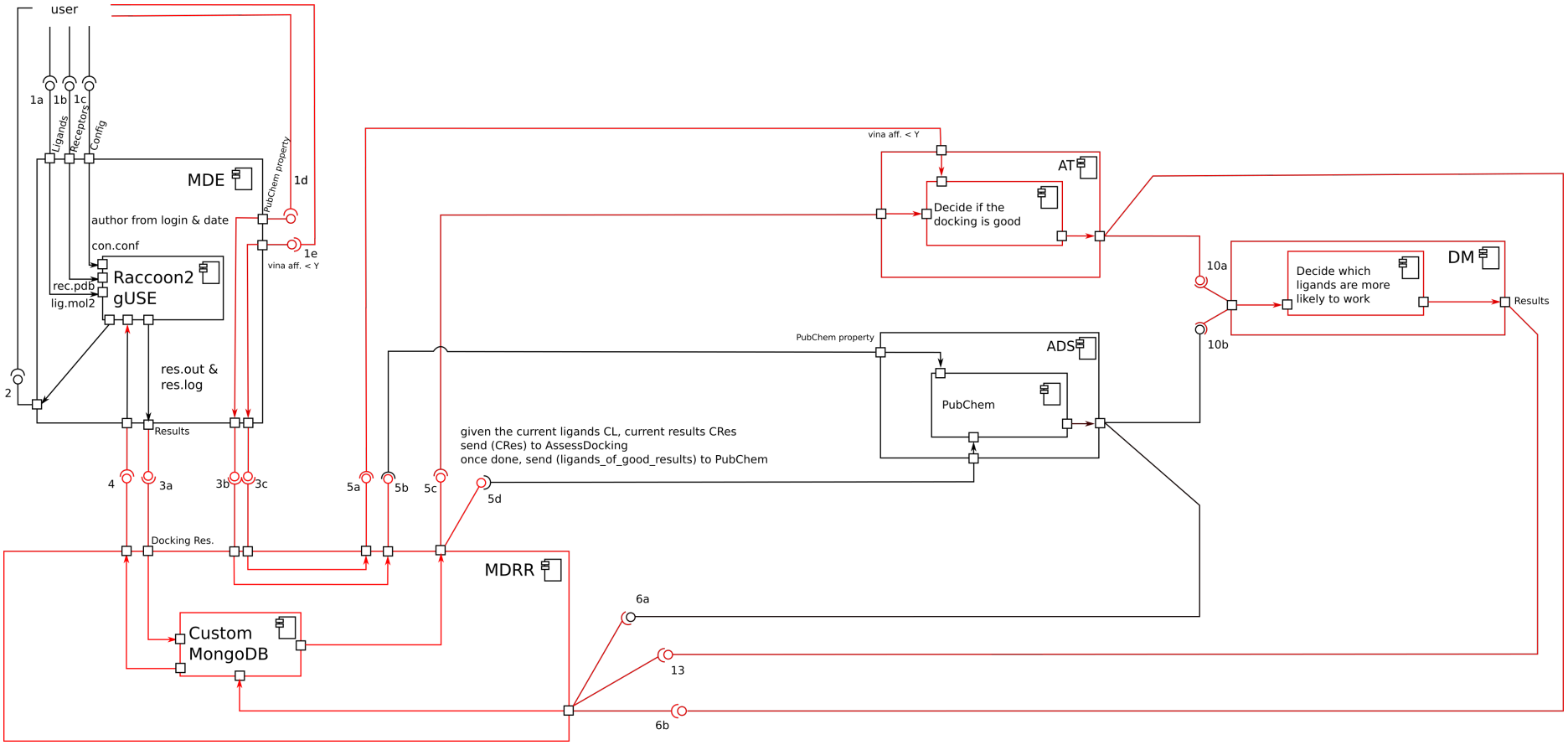

Figure 8.14: The detailed diagram of Scenario 2.

8.3.2 Code of Scenario 2

Similarly to Scenario 1, all the components in the implementation are accessible via a RESTful API developed using the Bottle web framework. Figure 8.15 provides more details about the flow and communication between the elements and the servers. The MDRR on Server 1 expects docking results from Raccoon2. It can store the results in the MongoDB database or continue with the scenario. The structure of the MongoDB is the same as in Sub-section 8.2.2.2.

The MDRR sends the results to the AT:AssessDocking on Server 2, along with the "threshold" value of AutoDock Vina affinity which is input by the user within Raccoon2. In the same manner as in Scenario 1, the AT:AssessDocking on Server 2 filters "good" docking results where the AutoDock Vina affinity is less than the threshold. When the docking assessment is completed, Server 1 receives a response and inserts a document in the analysis collection in order to keep track of the assessment of the results.

Following this, the MDRR selects the canonical SMILES codes of ligands that have been part of a "good" docking. If the docking results were inserted in the MongoDB database, the "canonical_SMILES" value can be selected from the database. However, Scenario 2 can be completed without storing the results in the database. In this case, the MDRR needs to calculate the canonical SMILES code based on the structure of the ligand (the .pdbqt file).

For each ligand, a request is sent to PubChem through the PUG-Rest API. The result from PubChem is the value of the property which has been provided by the user. This is then compared to the user-input threshold to filter the ligands. The DM on Server 1 in this scenario is minimal. A summary of the ligands that are part of a "good" docking and have the specified PubChem property within the threshold is sent to the MDE and displayed to the user. The analysis collection is updated with details about the decision as well as the previous steps. More details can be found in the source code on GitHub [237]. The remainder of this section will focus on how the coding step has used the abstract descriptions of elements and interfaces.

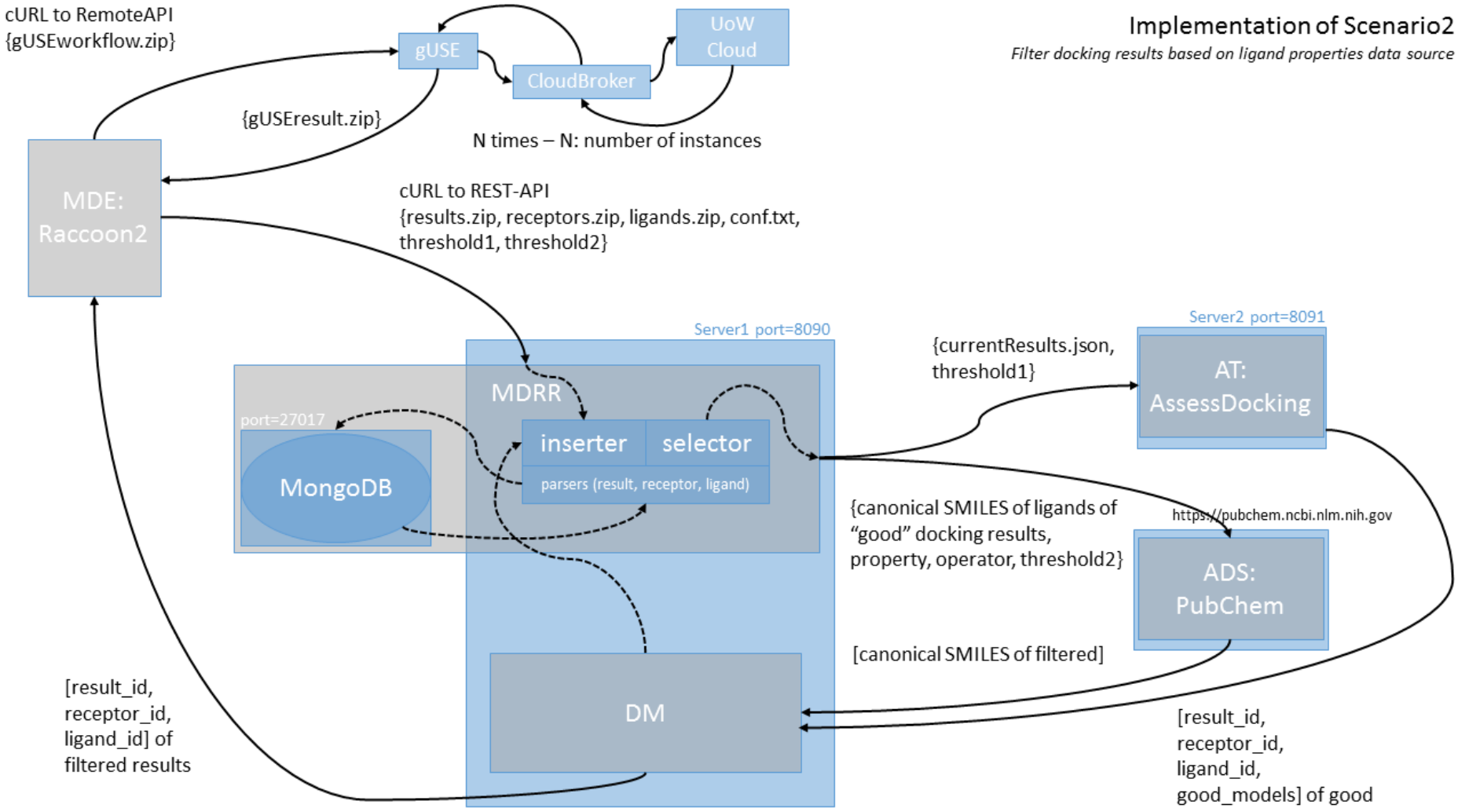

Figure 8.15: Communication between servers used in the implementation of Scenario 2.

8.3.2.1 Implementation of the MDE

The extended version of Raccoon2 (Chapter 3) should also ask the user for the values of an AutoDock Vina affinity threshold, a PubChem property name and threshold of the value of this property. It is also important to know whether the filtered results should be less than or equal to ($\leq$) or greater than ($>$) value. The need for these input fields can be derived from the detailed diagram (Figure 8.14).

8.3.2.2 Implementation of the MDRR

In Scenario 2, the current docking results are filtered based on a property of the ligands. The MDRR can store the docking results in the database, or continue without storing them. Otherwise, the MDRR and the MongoDB database are the same as in Scenario 1. The specifics of the database structure has been derived from the formal description as explained in Sub-section 8.2.2.2.

8.3.2.3 Implementation of AT:AssessDocking

One of the aims of implementing Scenario 2 (Section 8.1), was to show how an element can be reused. The source code shows that, once the docking results have been formatted correctly, the MDRR calls AT:AssessDocking in the same way as Scenario 1. The code of the AT was not altered at all.

8.3.2.4 Implementation of the ADS PubChem

Another aim of implementing this scenario, was to show how an element of a new element type, such as ADS, can be used. PubChem is an external element that has already been implemented by the National Center for Biotechnology Information. The code of Scenario 2 shows that within the MDRR, there should be a segment for obtaining and processing data from the ADS. The MDRR can use the PUG-Rest API provided by PubChem to obtain information regarding the ligand property. Due to the usage policy of PUG-Rest [242], a request is sent every 200 milliseconds. The fact that the MDRR uses an interface provided by the ADS is shown by the link MDRR – ADS in the detailed diagram (Figure 8.14).

8.3.2.5 Implementation of the DM

The DM in this scenario is minimal since the decision is merely the filtered ligands which have been part of a "good" docking result. This is shown in the formal description (Appendix D, page 182) where it is stated that *previous_result_filtered_ligands = previous_result_assessed_docking*. Similarly to Scenario 1, the result of the DM is returned to the MDE, and presented to the user (Figure 8.16).

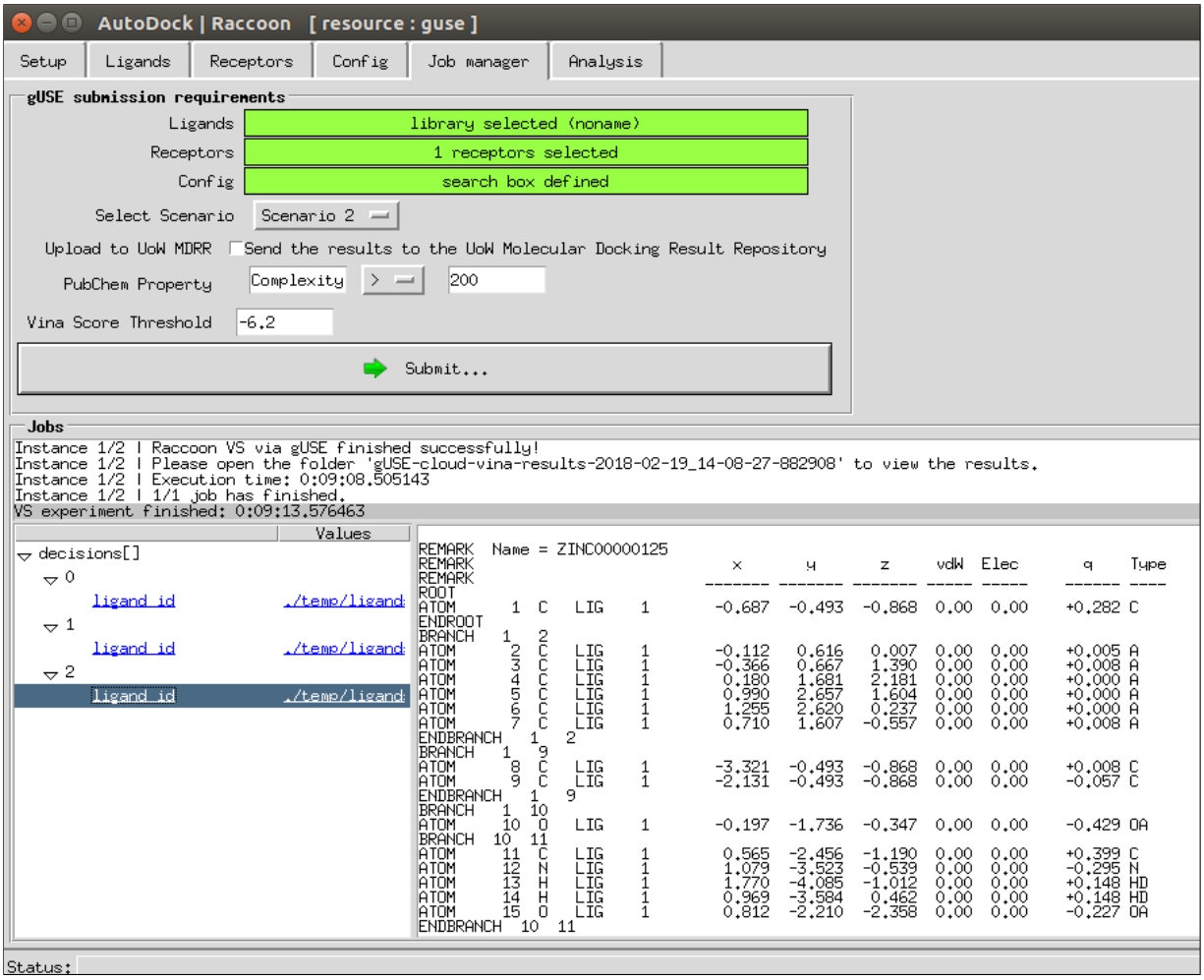

Figure 8.16: Screenshot of the final result of Scenario 2.

8.4 Implementing Scenario 4

The title of Scenario 4 is "Verify docking methodology and learn how to conduct docking by observing previous results with similar docking input". Scenario 4 starts with the user running a single docking simulation. Once completed, a similar receptor, ligand and configuration file to the ones used is returned. A basic diagram of Scenario 4 is shown in Figure 4.6. It proposes the use of eight elements of these four element types:

- MDE: The extension of Raccoon2 presented in this thesis.
- MDRR: A custom-made MongoDB-based repository.
- 5 × AT: the ligand similarity tool LIGSIFT, structural alignment tool DeepAlign, custom-made assessor of LIGSIFT, custom-made assessor of DeepAlign, and custom-made tool to compare config files.
- DM: a custom-made DM.

This diagram was derived from the basic diagram of the framework, so one can assume that Scenario 4 fits the framework. To provide a more precise analysis, the detailed diagram for each element and its interfaces is described in this section. A list of interfaces and a formal description can confirm that the proposed elements can be used as element types, prior to starting the coding step. This method is equivalent to the method used to describe Scenario 1 and Scenario 2. This section will focus on segments that are different and comment on the added value of implementing Scenario 4 using the methodology.

8.4.1 Abstract descriptions of Scenario 4

MDE: The extended version of Raccoon2 The MDE in Scenario 4 can be the same as the one in Scenario 1 (Section 8.2.1). Just like it was done in Scenario 2, the entire element can be reused with slightly different interfaces for the user-provided thresholds. Figure 8.21 shows that a detailed diagram of the Raccoon2 extension for Scenario 4 can be derived from the generic diagram of an MDE. When drawing this diagram, it is clear that the same element can be reused since most of the interfaces, and the core computation (the docking) are the same as in the detailed diagram of Raccoon2 used for Scenario 1.

Raccoon2 interfaces The same DeepScore threshold as in Scenario 1 should be provided as input, along with a threshold for the LIGSIFT score and the config comparison.

Formal description of Raccoon2 The formal description of the element Raccoon2 of Scenario 4 (Appendix E, page 185) can be derived from the formal description of the element type MDE. The result is the same as in Scenario 1 (Figure 8.3).

MDRR: a custom-made MongoDB Repository The same technique for searching a library of already implemented elements can be used to determine that Scenario 4 can reuse the same MDRR as in Scenario 1 (Section 8.2.1).

Interfaces of the custom-made MongoDB-based MDRR The interfaces of this MDRR are nearly the same as the interfaces of the MDRR in Scenario 1. Interfaces 5a-f and 6a-c are different in this scenario, as shown in Figure 8.21.

Formal description of the custom-made MongoDB-based MDRR2 When deriving the formal description of this MDRR (Appendix E, page 185), the same result as the MDRR of Scenario 1 is produced.

AT1: DeepAlign When using the technique for searching a library of already implemented elements (Technique 2 in Section 7.3), it can be seen that an already implemented AT that can be reused exists. At this point, there is a library with three implemented elements (AT:DeepAlign, AT:AssessDeepAlign, and AT:AssessDocking). Drawing the detailed diagram of the required AT shows that the interfaces and the core computation are the same as in the AT:DeepAlign and differ from the other two ATs. Therefore, the AT:DeepAlign (Section 8.2.1) will be reused. The same name for the AT can be used and the detailed diagram is nearly the same. The only difference is the naming of interface 5d (Figure 8.21).

Formal description of DeepAlign When deriving the formal description of this AT, the same methods as in Scenario 1 are created: *DeepAlignCore* and *DeepAlign* (Appendix E, page 188).

AT2: Assess DeepAlign results The same technique for searching a library of already implemented elements can be used to conclude that the AT:AssessDeepAlign from Scenario 1 (Section 8.2.1) can be reused. The same name for the AT can be used and the detailed diagram is nearly the same. The only difference is the naming of interface 6c (Figure 8.21).

Formal description of the custom-made threshold-based DeepAlign assessment AT When deriving the formal description of this AT, the same methods as in Scenario 1 are created: *goodDeepAlignResult* and *AssessDeepAlign* (Appendix E, page 188).

AT3: LIGSIFT The technique for searching a library of already implemented elements can be used for this AT as well. However, this search does not result in an element that can be reused. When drawing the detailed diagram for this AT, there are similarities between the interfaces of it and the AT:DeepAlign. However, the core computation is very different.

LIGSIFT calculates structural similarity of ligands, whereas DeepAlign calculates similarity of receptors. The remaining two ATs (AT:AssessDeepAlign and AT:AssessDocking) have more evident differences, as even the interfaces differ substantially. Therefore, the conclusion is that a custom-made AT should be created. The abstract description of this AT can be compared to the generic description of the element type AT to determine whether it fits the scenario. Figure 8.17 shows the detailed diagram which is similar to the detailed diagram of the element type AT.

Interfaces of LIGSIFT

5b. The LIGSIFT AT should require a LIGSIFT threshold.

5e. The LIGSIFT AT should require a list of ligands, and the target ligand from an MDRR.

7c-d. The LIGSIFT AT should provide the LIGSIFT threshold and the LIGSIFT result to an AT for assessment.

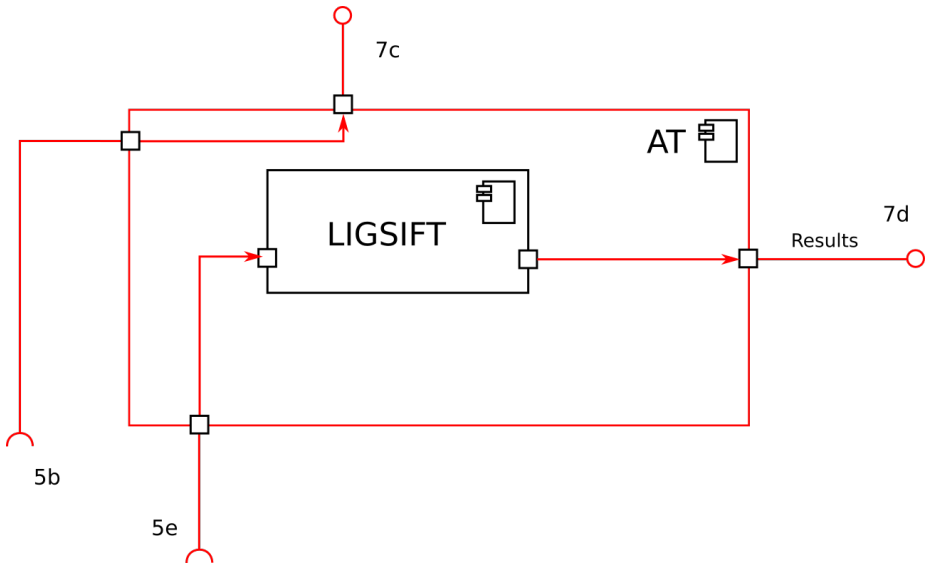

Figure 8.17: The diagram of the AT LIGSIFT.

Formal description of LIGSIFT One of the aims of implementing Scenario 4 is to show how a new element, which hasn't been used before, can be introduced (as mentioned in Section 8.1). This description is similar to the description of the AT:DeepAlign used in Scenario 1. This shows that if two ATs have similar interfaces and belong to the same class based on the type of input they receive, their formal descriptions can be created in an analogous manner. The abstract description of an AT in the framework included all possible classes of ATs based on the input they receive. The schema *LIGSIFT* (Appendix

E page 189) is derived from the schema *AdditionalTool* of the formal description of the element type AT (Appendix B, page 168). It specifies that the needed class of AT should be *LIGSIFTCore*, a tool that uses previous results, equivalent to *additionalTool_PR*. Specifically, this tool defines the input as a pair of ligands.

AT4: Assess LIGSIFT results The same technique for searching a library of already implemented elements can be used to determine that there is no element that can be reused. There are similarities between the interfaces of this AT and the AT:AssessDeepAlign, but the core computation is different. Similarly to the previous AT, the detailed diagram of a custom-made AT to assess LIGSIFT results can be compared to the diagram of the element type AT. Figure 8.18 confirms that the custom-made AT can be used since its diagram can be derived from the diagram of the element type AT.

Interfaces of the custom-made threshold-based LIGSIFT assessment AT

7c-d. This AT should require the LIGSIFT score threshold and the LIGSIFT result.

6a. This AT should provide the results of the assessment, whether the ligands are similar, to the MDRR for storage.

10b. This AT should provide the results of the assessment to the DM.

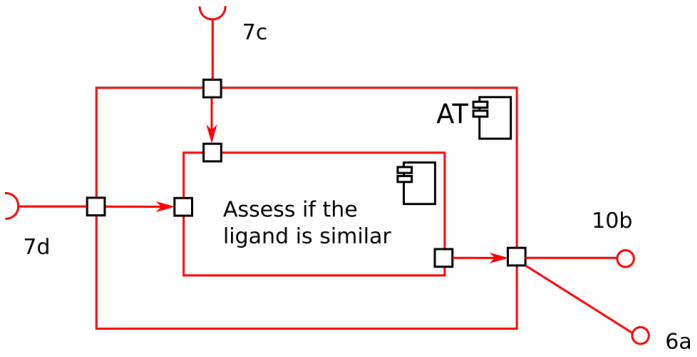

Figure 8.18: The diagram of the AT to assess LIGSIFT.

Formal description of the custom-made threshold-based LIGSIFT assessment AT The formal description of this AT is analogous to the formal description of the AT:AssessDeepAlign. The schema *AssessLIGSIFT* (Appendix E, page 189) is derived from the schema *Additional Tool*. The schema *goodLIGSIFTResult* expects the results of another AT and a user-provided input, which is equivalent to *additionalTool_UI_ATR* of the element type AT. The user-provided input needs to be a threshold used for comparison with the LIGSIFT score, as specified in *goodLIGSIFTResult*. The schema *As-*

sessLIGSIFTResult shows that a ligand will be considered similar to the target ligand only if the result of *goodLIGSIFTResult* is positive.

AT5: Custom-made tool to compare configuration files The same method as in the previous two ATs can help determine that there is no element that can be reused. The detailed diagram of the proposed custom-made tool for comparison of docking configuration files has similar interfaces as the AT:AssessDeepAlign and AT:AssessDocking, however it has a very different core computation. This diagram, shown in Figure 8.19, can be derived from the diagram of the element type AT, thus confirming that this element will fit the scenario.

Interfaces of the custom-made threshold-based configuration comparison AT

5c-f. This AT should require a configuration comparison threshold, and config files of past docking results along with a target config file.

6b. This AT should provide the results of the assessment (whether two configuration files are sufficiently similar) to the MDRR for storage.

10c. This AT should provide the results of the assessment to the DM for summarising.

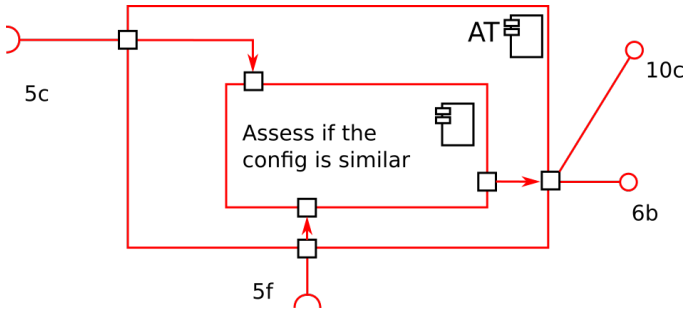

Figure 8.19: The diagram of the AT to compare configuration files.

Formal description of the custom-made tool to compare configuration files The formal description of this AT (Appendix E, page 190) resembles that of the AT:AssessDocking because they are the same class of AT based on the input. The method *similarConfig*, which is equivalent to the *additionalTool_UI_PR* of the framework's formal description, expects user-provided input and previous results as input. Specifically, it requires a threshold value and two configuration files. The schema *CompareToPreviousConfig*, which is derived from the schema *AdditionalTool*, shows that a positive result of *similarConfig* is required to consider the two config files "similar".

DM: custom-made element The DM combines the assessment from AT2 (if the receptors are similar), AT4 (if the ligands are similar), and AT5 (if the config files are similar). It should create a sorted list of similar previous docking results. It should be sorted firstly by similarity of receptor, then ligand, then config file. The fact that the detailed diagram of this custom-made tool (Figure 8.20) can be derived from the diagram of a DM element type of the framework, shows such a custom-made DM can be used in this scenario.

Interfaces of the custom-made DM

10a-c. This DM should require the results from AT2, AT4, and AT5.

13. The decision (in this case the sorted list of previous results with similar docking input) should be provided to the MDRR.

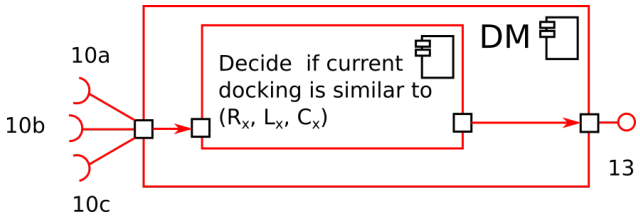

Figure 8.20: The diagram of the DM.

Formal description of the custom-made DM The schema *DecisionMaker_Custom* (Appendix E, page 190) is derived from the schema *DecisionMaker* of the generic element type DM (Appendix B, page 171). It uses *makeADecisionAdditionalTool* which requires the results of three ATs as input. In this scenario, the first result is a list of filtered receptors, the second a list of filtered ligands, and the third a list of filtered config files.

Detailed diagram of entire Scenario 4 The detailed diagram of the entire scenario can be created based on the analysis shown above (Figure 8.21). It confirms that Scenario 4 fits the framework since it can be derived from the detailed diagram of the framework. There are a total of 29 interfaces between these elements (numbered according to the numbering-scheme of the framework).

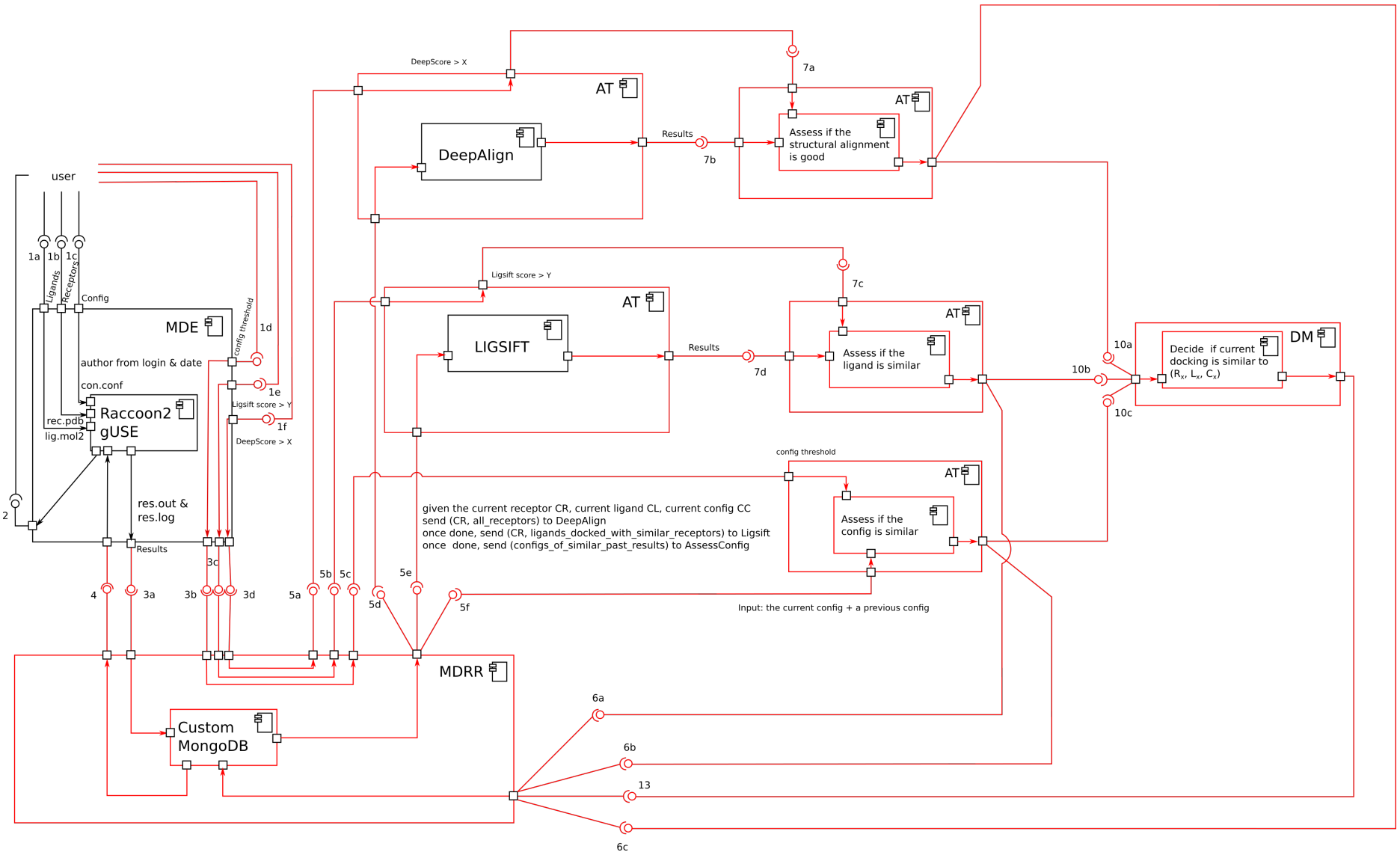

Figure 8.21: The detailed diagram of Scenario 4.

8.4.2 Code of Scenario 4

Similarly to Scenario 1 and 2, all the components in the implementation are accessible via a RESTful API developed using the Bottle web framework. Figure 8.22 provides more details about the flow and communication between the elements and the servers. In order to insert the results of the current VS simulation, the MDRR on Server 1 expects a zip file from Raccoon2. It inserts the results in the MongoDB database (which retains the same design as shown in Sub-section 8.2.2.2).

The goal of this scenario is to enable the user to verify the docking method or learn how to conduct docking simulations. To achieve this, the MDRR selects "all receptors" from the database and sends them to the AT:DeepAlign on Server 2, along with the "target receptor" (the receptor used in the original simulation), and a user-provided "threshold" value of DeepScore. The AT:DeepAlign on Server 2 executes DeepAlign for each receptor pair. It then calls the AT:AssessDeepAlign, located on the same server, to select "similar" receptors based on the DeepScore threshold. If the DeepAlign result is greater than the threshold, the two receptors are called "similar". A list of "similar" receptors is returned to Server 1 and used by the DM.

For each similar receptor, the MDRR selects only the ligands and configuration files of past docking results with that receptor. Then the MDRR completes two steps.

In the first step, it sends the ligands to be compared to the "target_ligand" (the ligand used in the current docking; If the user has conduced a VS, only the first ligand is chosen as target_ligand). The comparison is done by the AT:LIGSIFT and AT:AssessLIGSIFT. These ATs work in an analogous way to the receptor structural alignment tools mentioned above. Based on a user-provided threshold value, they determine if two ligands are similar.

In the second step, the MDRR sends all config files associated with each ligand in a previous docking for comparison with the "target_config". The comparison is done in the custom-made AT:CompareConfig. This AT compares the two configuration files geometrically and returns a list of similar config files based on a user-provided threshold. The three lists of similar receptors, ligands, and config files are processed by the DM. It returns one list with all the needed information ready to be visualised and presented to the user. The source code on GitHub [237] provides more information. The remainder of this section focuses on how the coding step has used the abstract descriptions of elements and interfaces.

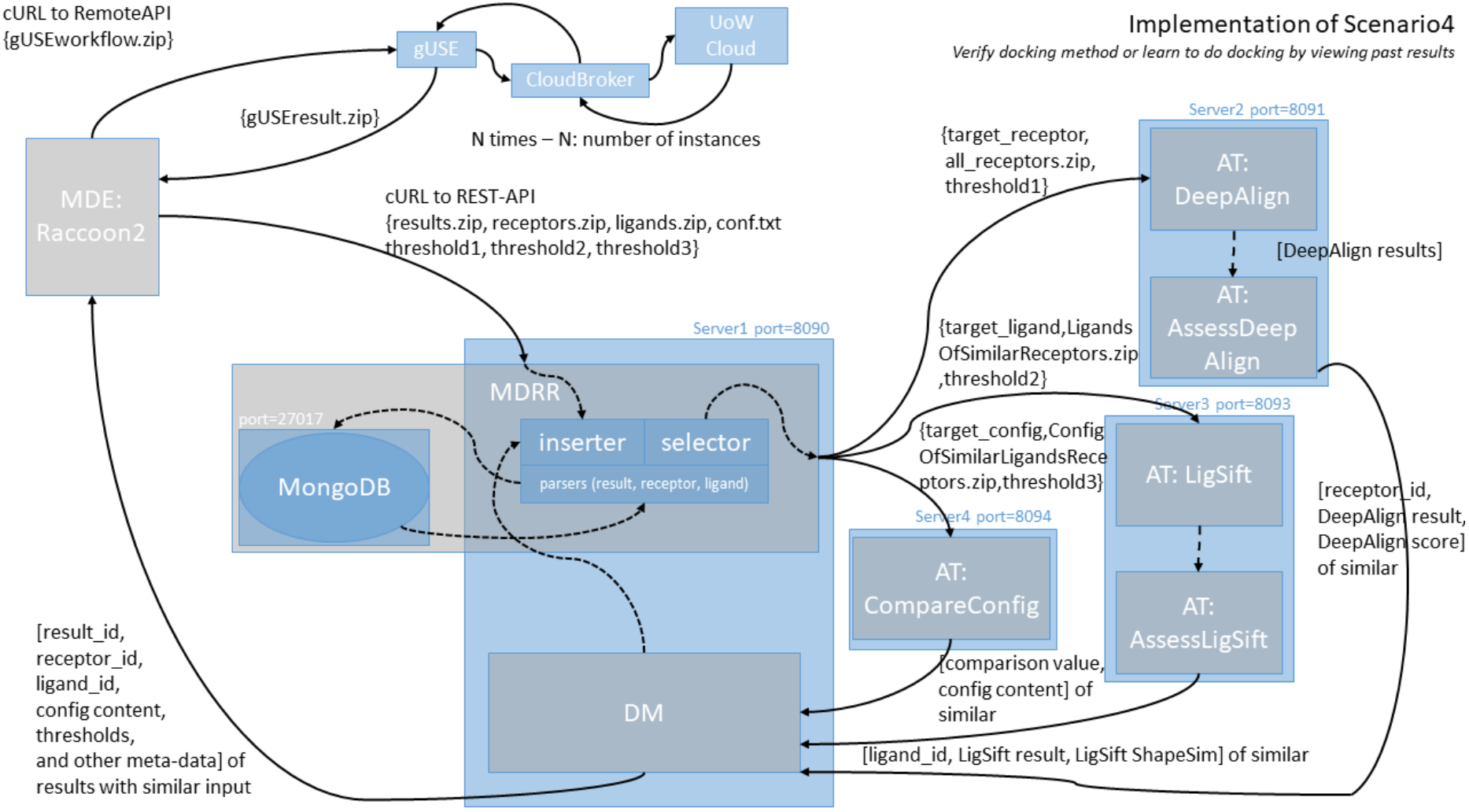

Figure 8.22: Communication between servers used in the implementation of Scenario 4.

8.4.2.1 Implementation of the MDE

Similarly to Scenario 2, the version of Raccoon2 (Chapter 3) can be reused with a slight modification. The user should be able to enter the required threshold values for: DeepScore, the LIGSIFT score, and the configuration comparison score. These required user inputs are shown in the detailed diagram (Figure 8.21).

8.4.2.2 Implementation of the MDRR

The custom-made MDRR can also be reused. The code needs to be updated to handle the flow required for Scenario 4 and process the three threshold values and the docking results received from Raccoon2 as it is specified in Figure 8.21.

8.4.2.3 Implementation of the ATs

AT:DeepAlign and AT:AssessDeepAlign The implementation of Scenario 2 showed how an element can be reused. The same technique for reusing elements is applied in Scenario 4. The MDRR needs to correctly format the input for the AT:DeepAlign, but the code of the ATs themselves has not been changed.

AT:LIGSIFT and AT:AssessLIGSIFT The implementation of these two ATs is analogous to the AT:DeepAlign and AT:AssessDeepAlign. This can be seen in the detailed diagram and the formal description. The interfaces of AT:LIGSIFT and AT:DeepAlign are similar, only the core computation is different. There is an analogy between the way that similar receptors are filtered and the way that similar ligands are filtered. Therefore, in this implementation, a new thread is launched for each ligand pair in order to run the LIGSIFT executable and send the results to be assessed.

The value of "ShapeSim" calculated by LIGSIFT has been chosen as a representative LIGSIFT score which is compared to the user-provide threshold in the AT:AssessLIGSIFT. ShapeSim is the shape-based scaled TC, known as sTC (more details are provided in Section 2.3.2). If the ShapeSim value is greater than the threshold, the two ligands are considered "similar".

AT:CompareConfig Even though the description of this AT resembles that of AT:AssessDocking and AT:AssessDeepAlign, they cannot be reused because their core computations are not comparing docking configuration files. Some of the concepts used in

the code of these tools can be seen in the code of the custom-made AT:CompareConfig. After processing the received input, this AT compares the currently used configuration file with each of the configuration files provided. This thesis has mainly used AutoDock Vina as an example docking tool, thus this AT was developed with an AutoDock Vina configuration file in mind. An AutoDock Vina configuration file contains several parameters including the coordinates of a cuboid where the docking calculations will be focused. Usually, this cuboid is positioned around the potential binding site of the receptor. The cuboid is described with two 3-dimensional points: the centre of the cuboid, and the "size" of the cuboid represented by the coordinates of one of the vertices, relative to the centre. The "size" is equivalent to one half of the size of the sides if the centre were at (0, 0, 0) as shown in Figure 8.23. Because it aims to describe the size, the values will always be positive, so it is the particular vertex where $x > 0$, $y > 0$, and $z > 0$. In order to calculate a similarity value, distances between the two 3-dimensional points are calculated (Equations 8.1 and 8.2).

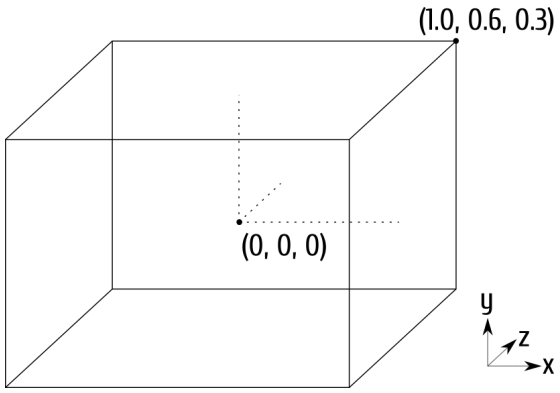

Figure 8.23: Representation of the cuboid of an AutoDock Vina configuration file.

$$DistanceC = \sqrt{(C_x - TargetC_x)^2 + (C_y - TargetC_y)^2 + (C_z - TargetC_z)^2} \tag{8.1}$$

and

$$DistanceS = \sqrt{(S_x - TargetS_x)^2 + (S_y - TargetS_y)^2 + (S_z - TargetS_z)^2} \tag{8.2}$$

where C_x, C_y, C_z are the coordinates of the centre of a cuboid from a candidate configuration file; $TargetC_x$, $TargetC_y$, $TargetC_z$ are the analogous coordinates from the file originally used by the user; S_x, S_y, S_z are the coordinates of the mentioned vertex representing the size of the cuboid; and $TargetS_x$, $TargetS_y$, $TargetS_z$ are the analogous coordinates from the file originally used by the user.

The mean of the two distances ($\frac{DistanceC + DistanceS}{2}$) is taken as the comparison value. If this comparison value is less than the user-provided threshold, then the two cuboids, and ultimately, the configuration files are deemed "similar". The conclusion of the comparison is strictly geometrical, and may not always be biologically relevant. Furthermore, the user would have to be acquainted with the method to be able to provide a correct threshold.

8.4.2.4 Implementation of the DM

The DM of Scenario 4 receives the assessed similar receptors, ligands, and config files (seen in the detailed diagram, Figure 8.20). It groups them in a list where the list item is a ligand–receptor–config triplet. After it adds meta-data (e.g. the similarity scores), the DM sorts this list. The list is sorted firstly based on the receptor similarity, then based on the ligand similarity value. Finally, the list is returned to the MDE and visualised. Figure 8.24 is a screenshot showing the final result which is displayed to the user in the GUI of Raccoon2.

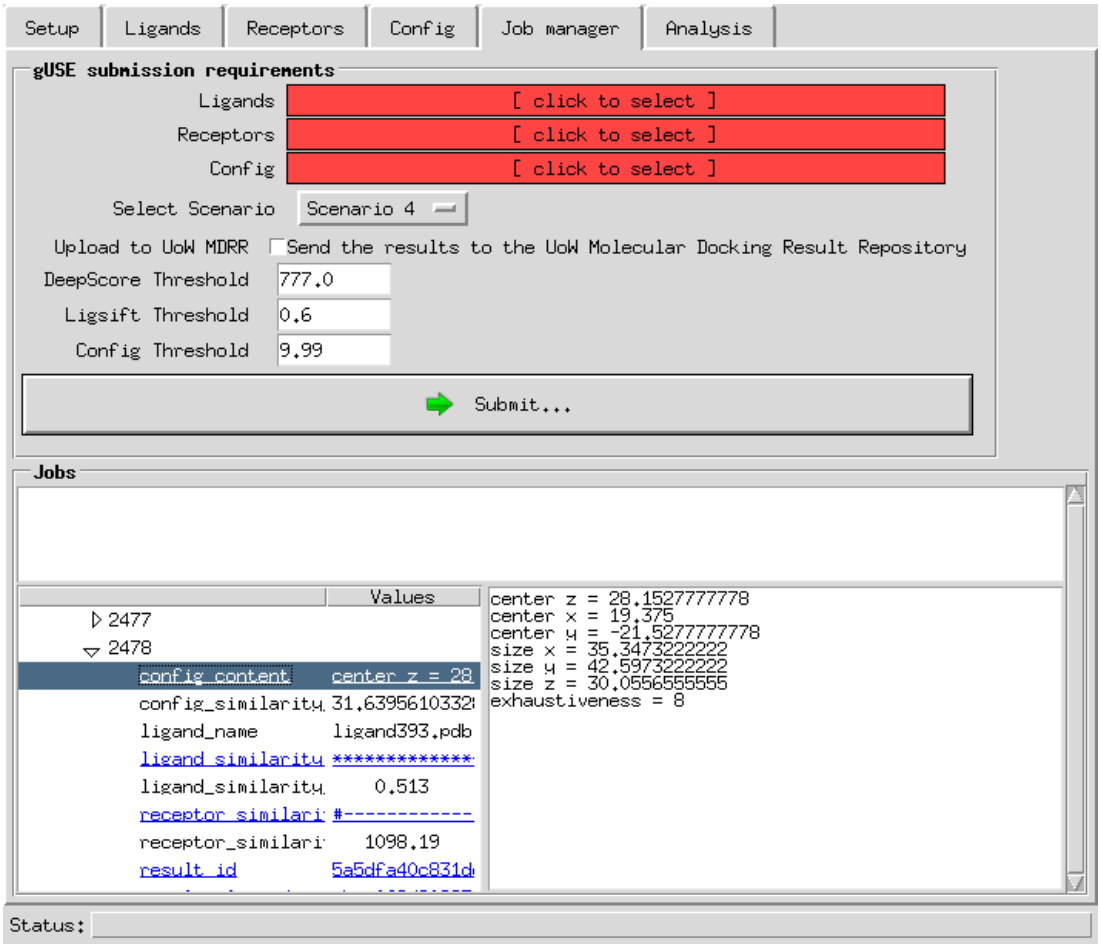

Figure 8.24: Screenshot of the final result of Scenario 4.

8.5 Conclusion

This chapter evaluated the framework and methodology by providing a detailed description of how three scenarios can be implemented. The scenarios were firstly described using the abstract (diagrammatic, textual, and formal) description, then a prototype implementation was produced and outlined. The goal of the implementations was to show how the

framework and methodology can be used by the software development team to produce a viable software system. All three implementations showed how developers can use the three techniques of the methodology (Section 7.3) to develop software using a structured and methodical approach. The approach includes a technique to determine whether a new scenario would fit the framework. It also includes a technique to search a library of abstract descriptions for an already implemented element that can be reused. Finally, it includes a technique to determine whether a newly proposed custom-made tool can be used in the implementation. This shows that using the framework and methodology provides an approach that is beneficial for software developers. The following chapter examines the usability of the implementations from the point of view of the users - the biomedical scientists.

Chapter 9

Usability of Implementations

9.1 Introduction

The benefits of implementing the scenarios using a methodical approach, such as the abstract descriptions specified in the framework, were outlined above. In particular, following the methodology allows the development team to determine: whether a scenario fits the framework, whether already implemented elements can be reused, or whether a proposed new element fits the appropriate element type (Section 7.3). These can be seen as benefits for the software developers.

However, it is possible that following such a methodical approach produces more cumbersome and less usable systems. This section will show that this is not the case for the three scenarios implemented following the methodology and framework. Completing a scenario using currently available tools directly (without the implementation) will be compared to completing the scenario "with" the implementation. The fact that using the implementation is at least as usable as the alternative shows that the framework and methodology produce usable systems.

The candidate conducted all the usability tests, with guidance from the two life scientists (mentioned in Section 7.4) who confirmed that the final result of the three scenarios were useful from a biomedical point of view. The implementations of the three scenarios implemented using the framework were demonstrated to the life scientists during an internal mini-workshop. The life scientists were shown how the implementations worked, and they were guided through the process of obtaining the final results. Finally, the results were analysed and discussed.

9.2 Planning the Usability Tests

Planning usability tests of Scenario 1 In this scenario the user completes a docking or VS. Then, in search of a suggestion of what to dock to the target receptor in the future, the user looks for similar receptors. If some ligands have been docked to these similar receptors, and the docking is considered to be "good", then these ligands are a good suggestion for the next docking with the target receptor. The steps needed to complete this scenario with and without the implementation are depicted in the flowcharts in Figure 9.1.

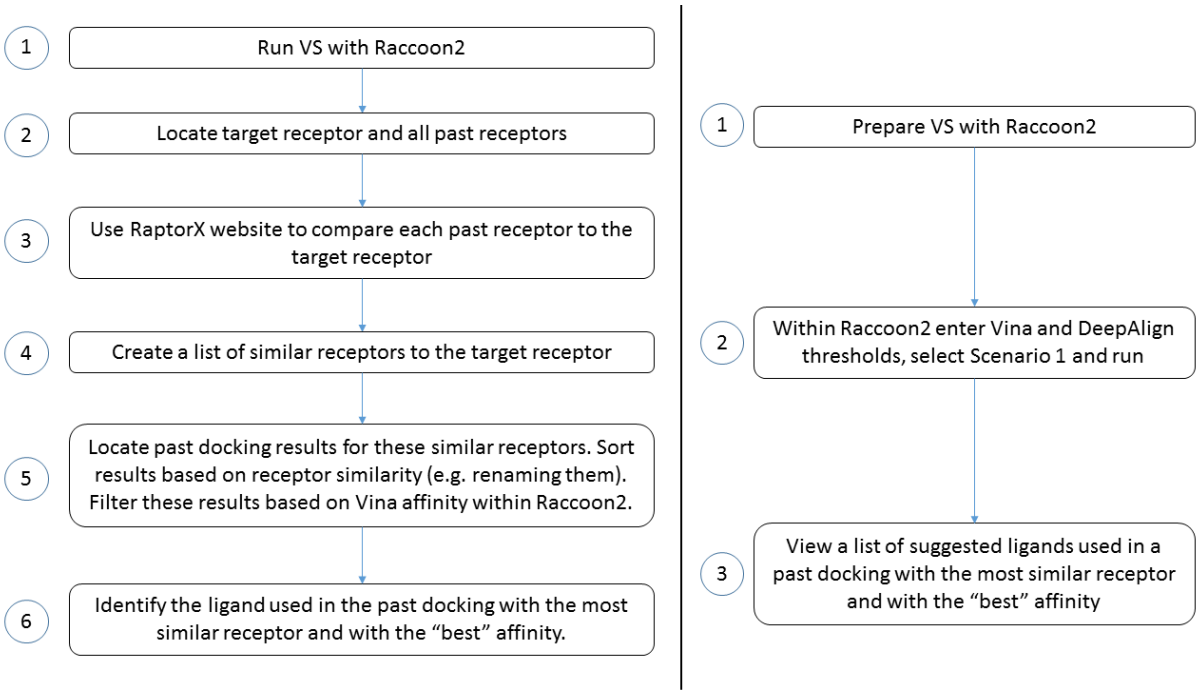

Figure 9.1: Flow of events of Scenario 1 "without" the implementation (left) vs. "with" the implementation (right).

When conducting the tests, the tool AutoDock Vina was considered. Some details may be different if a different docking tool is used. Each usability test may require an "Assumption" that something is done as a pre-requisite before starting the user-test, and several main "Process" steps. The following list contains the steps needed to complete this scenario "without the implementation".

Assumption: Scientist has stored previous docking results.

Process:

1. Run VS with Raccoon2. The results can be viewed in Raccoon2, but at this point the user requires a suggestion for the next docking.

2. Locate the target receptor in the file system. Locate all the receptors that are part of the stored previous docking results.

3. Open the RaptorX website in the browser and manually upload the target receptor and another receptor that has been used in the past. Do this for each receptor used in the past. RaptorX allows uploading batches of receptors (the limit is 25) which may be used instead of uploading them one by one.

4. Once the RaptorX Structural Alignment website returns a result for all the receptors, a list of the similar receptors to the target receptor can be created.

5. Locate the past docking results that you have stored for each of the similar receptors. Raccoon2 can be used to filter these results. The docking results can be uploaded in the "Analysis" tab and then filtered based on the AutoDock Vina affinity. The docking results should be sorted by the receptor similarity value or the files could be renamed to include this value. A script may be written to do this in case of a large number of receptors. Then, the results should be uploaded separately for each receptor.

6. Identify the past docking result of the most similar receptor with the "best" affinity. The ligand used in this docking is the suggested ligand to dock to the target receptor next.

The usability test "with" the implementation can be written in the same format. This usability test claims certain benefits for the user ("Claim").

Assumption: The MDRR has enough relevant docking results for this scenario to produce meaningful results.

Process:

1. Prepare VS with Raccoon2. Before running it, additional information can be added in order to obtain a suggestion for the next docking.

2. Within the Raccoon2 GUI, in the "Job manager" tab, the implementation has provided a direct way to conduct Scenario 1. The AutoDock Vina affinity and DeepScore thresholds can be entered in the provided text fields. Clicking "Submit" will run the VS and look for a suggestion of the next docking.

3. Once both of these actions are completed, the VS results and more importantly, a list of suggestions for the next docking, can be viewed in Raccoon2. The list is sorted by the most similar receptor first, and then if there are several ligands per receptor they are sorted by the AutoDock Vina affinity of the respective docking result.

Claim: The user will require less expertise and less time to conduct this scenario because most manual steps are automatic.

Planning usability tests of Scenario 2 In this scenario the user completes a VS simulation (docking a large number of ligands and one receptor). Then, the docking results need to be filtered based on properties of the ligands, specifically, molecular properties that are stored in external data sources such as PubChem. These filtered results would assist the scientist in making a conclusion and increase the likelihood of showing that the two molecules bind in the wet lab. The steps needed to complete this scenario with and without the implementation are shown in Figure 9.2.

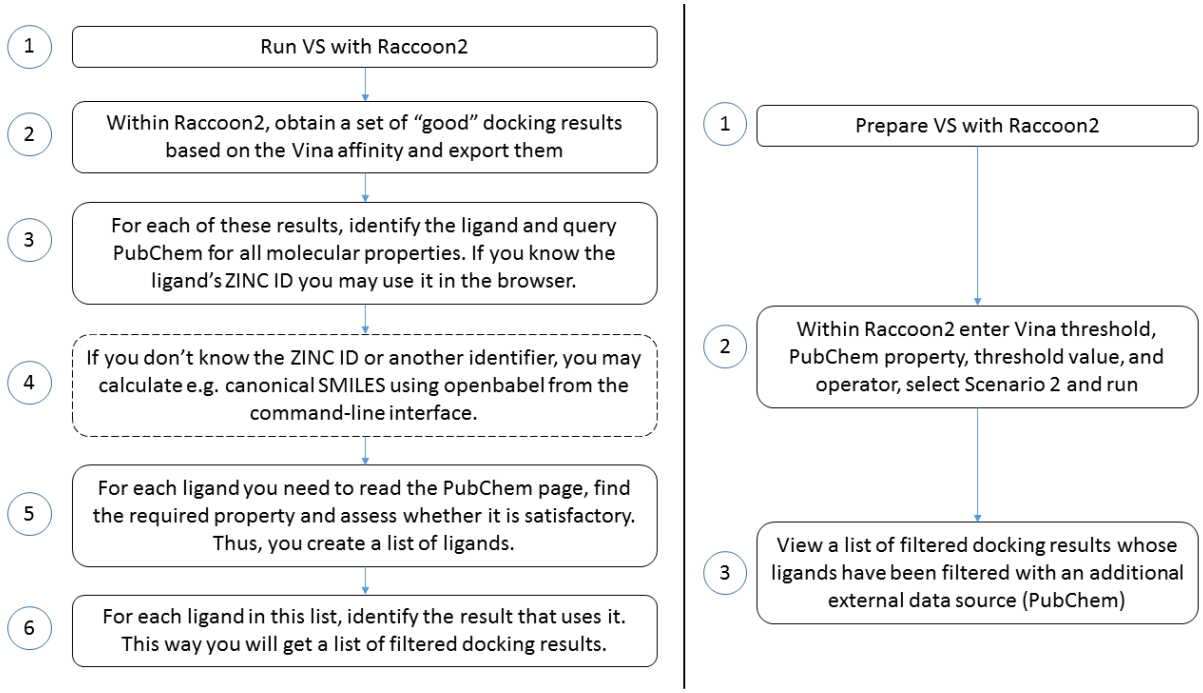

Figure 9.2: Flow of events of Scenario 2 "without" the implementation (left) vs. "with" the implementation (right).

The following list contains the steps needed to complete this scenario "without the implementation" in the same format.

Assumption: None

Process:

1. Run VS with Raccoon2. The results can be viewed in Raccoon2.

2. A set of "good" docking results based on the Vina affinity can be obtained within the "Analysis" tab of Raccoon2.

3. Each of these results contains a ligand. The PubChem Compounds database can be queried for each of these ligands through the browser. If the ZINC ID or another identifier for each ligand is known, then it can be entered in a text field.

4. Optionally, if it is not known, an identifier can be calculated. This can be done using

the openbabel tool from the command-line interface, for instance, to calculate the canonical SMILES value from the structure of the ligand. The command for one molecule is "`obabel -ipdbqt name_of_ligand.pdbqt -osmi`", and a batch calculation can be done with "`obabel *.pdbqt -osmi -m`". These commands assume the structure file format of the ligands is .pdbqt.

5. The PubChem page can be viewed for each ligand and the particular property can be manually identified. Whether its value satisfies a criteria can be observed. If it does, that ligand becomes a member of a list of filtered ligands.

6. For each ligand in this list, identify the docking result that uses it. That docking result will be part of a list of filtered results.

The following list contains the steps needed to complete this scenario "with the implementation" in the usual format.

Assumption: None

Process:

1. Prepare the VS simulation with Raccoon2. Before running it, additional information can be added in order to obtain a filtered list of the results based on ligand properties.

2. Within the Raccoon2 GUI, in the "Job manager" tab, the implementation has provided a direct way to conduct Scenario 2 and enter several inputs. The AutoDock Vina affinity threshold can be entered in a text field. The name of the PubChem property, its threshold value, and whether the threshold represents the minimum or maximum value for the filtered ligands.

3. Once these actions are completed, the VS results can be viewed in Raccoon2 and more importantly, a filtered list of the results that have a ligand whose chosen PubChem property fits the criteria can be observed within Raccoon2.

Claim: The user will require less expertise and less time to conduct this scenario because most manual steps are automatic.

Planning usability tests of Scenario 4 In this scenario the user completes a single docking simulation (in case of a VS, the first receptor, ligand, and config will be considered). Then, the user observes other docking results with similar input files in order to

either verify that the way that the docking has been conducted is correct, or in the case of a novice user, to learn how to conduct docking correctly. The steps needed to complete this scenario "with" and "without" the implementation are shown in Figure 9.3.

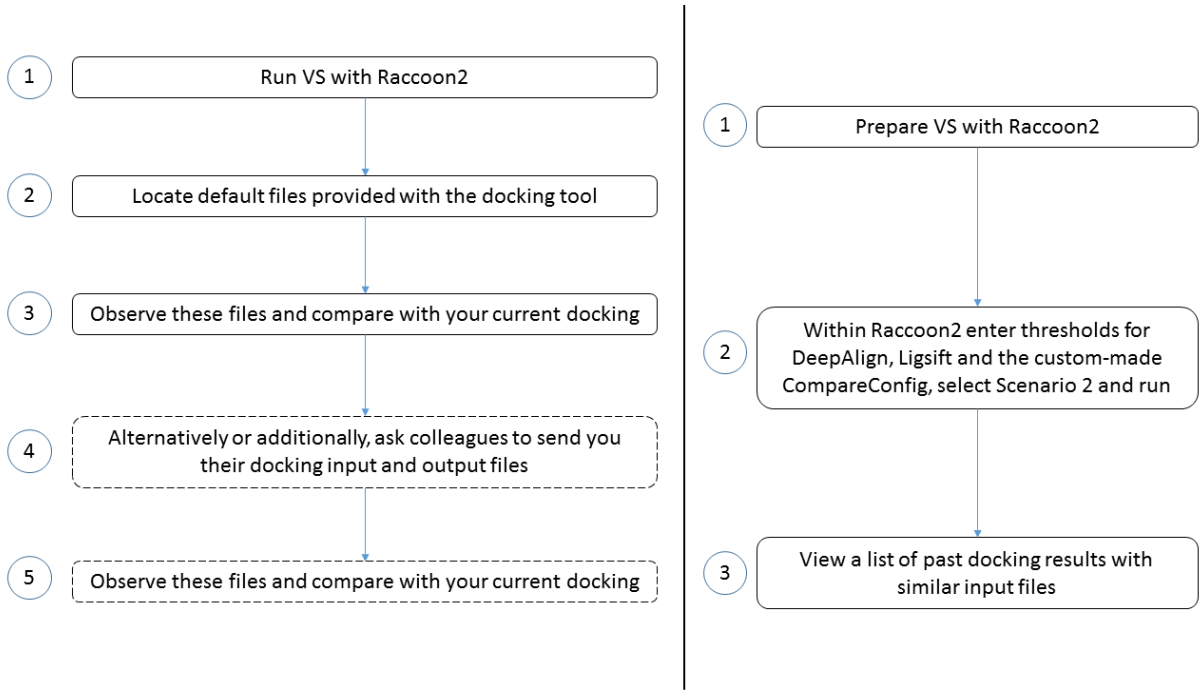

Figure 9.3: Flow of events of Scenario 4 "without" the implementation (left) vs. "with" the implementation (right).

The following list contains the steps needed to complete this scenario "without the implementation" in the format that includes prior assumptions and main process steps.

Assumption: There are default docking input and output files provided as part of the docking tool, or input and output files of properly conducted docking can be acquired from other users.

Process:

1. Run a docking simulation with Raccoon2. The results can be viewed in Raccoon2.

2. Locate the default input files provided with the docking tool.

3. A potential mistake in the way the docking has been conducted can be identified, if there is a big difference. Or, if the user is a novice, these files can be studied in order to learn how to conduct docking.

4. Another user can be asked for input and output files of their docking.

5. These files can be used to identify a potential mistake in the way the docking has been conducted, or if the user is a novice, they can be studied in order to learn how to conduct

docking.

The following list contains the steps needed to complete this scenario "with the implementation" in the familiar format.

Assumption: The MDRR has enough relevant docking results for this scenario to produce meaningful results.

Process:

1. Prepare a docking simulation with Raccoon2. Before viewing the results, additional information can be provided in order to obtain a list of previous docking results that have similar input files to the docking simulation.

2. Within the Raccoon2 GUI, in the "Job manager" tab, the implementation has provided a direct way to conduct Scenario 4. Threshold values can be entered for the receptor similarity tool DeepAlign, ligand similarity tool LIGSIFT, and the custom-made tool for comparing configuration files. Then, clicking on "Submit" will both run the docking and look for past docking results with similar input files.

3. Once completed, the docking results can be viewed in Raccoon2. More importantly, a list of past docking results with similar input files with more details about the results themselves, or each of the input files (receptor, ligand, config) can be analysed.

Claim: The user will require less expertise and less time to conduct this scenario because most manual steps are automatic. Additionally, the implementation removes the need to rely on default files provided with the docking tool, or receiving files from colleagues.

9.3 Preparation of usability tests

As emphasised by the plans for the tests, particularly the assumptions for the tests of Scenarios 1 and 4, there is a need for an MDRR which contains relevant previous docking results. In order to achieve this, a total of 166,320 docking simulations were conducted and their results stored in the database. The extended version of Raccoon2 was used to produce these docking results, and they were stored directly into the MDRR by utilising the interface created for the implementation of the scenarios. The UoW academic cloud was used as the DCI to run the docking simulations.

The same receptor used in the proof-of-concept test of the extension of Raccoon2 in Chapter 3 was used as a target receptor. It represents the ribokinase of *Trichimonas vaginalis*

(TV). When choosing which receptors to dock in order to fill the MDRR, a number of *a priori* similar receptors to the TV ribokinase were chosen. After searching the wwPDB, a total of 23 solved protein structures of ribokinase were found. These came from 7 different species[1]. Some species had more than one version of this protein. Only one was chosen for each species[2], thus selecting 7 *a priori* similar receptors. These are structures of the same type of protein, so they are likely to have the same ancestor and they are homologous. To check if there is structural alignment, DeepAlign was run between the TV ribokinase and each of the 7 ribokinases. The results showed a high DeepScore value (min: 975.47 for 3I3Y, max: 1491.79 for 5BYD).

The next step requires a number of random receptors which may or may not be similar to the TV ribokinase. In order to obtain these, the RCSB "Molecule of the Month" series [243] was used. When selecting molecules, the first molecule mentioned in the article for a particular month was manually chosen. This molecule was downloaded and converted to .pdbqt. If the conversion failed, the molecule was discarded, and the molecule mentioned next was selected. Following this, a test AutoDock Vina docking was run. If the test docking returned errors, this molecule was discarded, and the molecule mentioned next was selected. If the docking started correctly, that molecule was chosen as one of the randomly selected receptors. A total of 63 receptors were chosen this way. Thus, the set of receptors had 70 members, 7 of them (10%) *a priori* similar to the TV ribokinase, and 63 (90%) other random receptors.

Furthermore, a large number of ligands were needed. A set of molecules that have been approved as drugs in some jurisdiction in the world was identified from ZINC [244]. A total of 2376 such molecules were downloaded and saved as individual .mol2 files.

Finally, each receptor requires its own configuration file. The configuration files were created within the GUI of Raccoon2. Care was taken for the cuboid to cover a part of the receptor. Further analysis to see where the receptor has an active site, or a biologically relevant part, were not conducted. This procedure was followed for each of the receptor, thus creating 70 configuration files.

These input files were used to fill the MDRR with a large number of docking results. The final number of docking results in the MDRR was 166,320 (70×2376). A total of 80 runs of Raccoon2 were conducted, one for each receptor (10 had to be repeated due to faults or issues with the infrastructure). Most of the runs used 3 or 6 instances in the UoW cloud. The UoW cloud executed a total of 393 jobs for this exercise, with an average execution time of *2h 23min 23s*.

[1] *Escherichia coli*, *Homo sapiens*, *Klebsiella pneumoniae*, *Mycobacterium tuberculosis*, *Staphylococcus aureus*, *Thermotoga maritima*, and *Vibrio cholerae*.

[2] PDB IDs: 1RKA, 5BYD, 3I3Y, 3GO6, 3RY7, 1VM7, and 4X8F respectively.

9.4 Results of usability tests

In order to comment on the claimed benefits of the implementations, three scenarios "with" and "without" the implementation were conducted, resulting in 6 tests. The focus of these tests is the additional analysis provided as a result of the three scenarios, and not the results of the docking simulations themselves. In all of the tests, the original docking (or VS simulation) used the TV ribokinase as a receptor, an adequate configuration file, and a group of 10 ligands. These 10 ligands were the first 10 of the 130,216 ligands used in the proof-of-concept of the extended Raccoon2 (Chapter 3). In the following paragraphs the test results will be described. The numbering of events corresponds to the "Process" steps of the usability test plans.

Results of usability tests of Scenario 1

Results of usability tests "without" the implementation:

1. The VS (10 ligands and 1 receptor) with Raccoon2 on the cloud finished in 10 minutes.

2. The target receptor can be easily located on the file system, since this location was used to upload the target receptor to Raccoon2 when preparing the VS. The locations of the past results can be a lot more difficult to find. In this case, the group of results used to fill the MDRR was selected. These were easy to locate as they have been stored in a relatively well-designed folder structure.

3. The user interface of the RaptorX Structural Alignment website was found to be intuitive. The user can upload two receptors and proceed with pair-wise structural alignment. The DeepAlign results are visible on the page after about 1 minute. The results contain several similarity measures, but the DeepScore value is not visible. This is unusual since DeepScore is the main score calculated by DeepAlign.

4. The more receptors are found to be similar, the more cumbersome it is to create a list of similar receptors.

5. Results of 1 similar receptor were used, so there was no concern about the order of results being based on the similarity value of the receptors. Renaming results to contain a DeepAlign similarity value would require writing a small script that would select the correct similarity value for each receptor and rename the name of the result file that has used this receptor. If the docking result names contain the name of the receptor at the start, this script will be simpler. The remaining part of this step refers to filtering results with Raccoon2. The "Analysis" tab of Raccoon2 is very intuitive and it took nearly 8 minutes to filter a set of 2376 results for 1 receptor. If this is done for the results of each receptor it will get very cumbersome.

6. If the results are filtered per receptor, and ordered according to the receptor similarity, then the results of the most similar receptor can be easily identified. If there are more than one docking results for the most similar receptor, Raccoon2 can be used. The results can be uploaded to Raccoon2 and they will be sorted based on the AutoDock Vina affinity.

Results of usability tests "with" the implementation:

1. The VS in Raccoon2 is prepared the standard way.

2. The implementation provides two text fields to enter the AutoDock Vina affinity and DeepScore thresholds. The user would require some knowledge of the meaning of these values. The implementation provides the default values of "-6.8" for the AutoDock Vina affinity, and "777.0" for DeepScore.

3. After less than 30 minutes, the user can view the results of Scenario 1 within the "Job manager" tab. Note that the usability test was ran on an 8GB RAM, $4 \times 2.50GHz$ CPU computer. The performance would be better if the implementation was deployed on a more optimal infrastructure. The tab that shows the results of this scenario includes a "TreeView" which can be expanded, and a second pane which is filled with additional information upon clicking the ID of a ligand, receptor, or result (Figure 8.11).

Results of usability tests of Scenario 2

Results of usability tests "without" the implementation:

1. The VS (10 ligands and 1 receptor) with Raccoon2 on the cloud finished in 10 minutes.

2. The "Analysis" tab in Raccoon2 provides a simple way to filter docking results based on the AutoDock Vina affinity. The user can specify a range, e.g. [-5.5, -6.5].

3. In this test, the ZINC ID of the molecule was part of the ligand's name. For a small amount of ligands it was easy to copy and paste the ZINC ID into a text field on the PubChem website (specifically the web interface of the "Compounds" database).

4. Because the ZINC ID is known, this step was not necessary.

5. The chosen property is "Complexity", which cannot be easily pre-calculated. The name of this property can be used to search the browser page describing the molecule. For a large set of ligands it would be too cumbersome and time-consuming to do this manually.

6. By default the docking result in Raccoon2 is named "receptor-name_ligand-name_out.pdbqt" which makes identifying which ligand was used easy. However, for a large number of ligands, doing this manually without an additional script can be very error-prone. Particularly when the ligand names are ZINC IDs which are series of numbers.

Results of usability tests "with" the implementation:

1. The preparation of the VS in Raccoon2 is the same as without the implementation.

2. The implementation provides simple text fields to enter the AutoDock Vina affinity and the PubChem property input including a value from a drop-down list containing "$\leq$" and "$>$". The values "`AutoDock Vina threshold = -6.2`" and "`Complexity > 200`" were used in this usability test.

3. After 10 minutes, the user can view the results of Scenario 2 within the "Job manager" tab. The tab that shows the results of this scenario includes a TreeView which can be expanded, and a second pane which is filled with additional information upon clicking the ID of a ligand, receptor, or result (Figure 8.16).

Results of usability tests of Scenario 4

Results of usability tests "without" the implementation:

1. The VS (10 ligands and 1 receptor) with Raccoon2 on the cloud finished in 10 minutes.

2. Default docking input files are provided with AutoDock Vina [245] along with a video that, once followed, will produce output files.

3. Following this video and examining the docking files is a good way to verify one's docking method as well as learn how to conduct docking. However, the particular video seems outdated. It requires users to include the parameter "`all=all.pdbqt`" in the configuration file, although this will cause an error with the latest version of AutoDock Vina. Perhaps as a result of this, the output files obtained while following the video in this usability test differ slightly from the output files shown in the video.

4. This test did not use docking files obtained from other users.

5. This test did not use docking files obtained from other users.

Results of usability tests "with" the implementation:

1. The preparation of the VS in Raccoon2 is the same as without the implementation.

2. The implementation provides three text fields to enter the thresholds for DeepScore, LIGSIFT, and the custom-made CompareConfig tools. The user would have to have some knowledge of the meaning of the DeepScore, LIGSIFT's ShapeSim, and value used in the the config comparison tool. The default values of the implementations ("777.0", "0.6", and "9.99" respectively) were used.

3. After 2 hours and 18 minutes, the user can view the results of Scenario 4 within the "Job manager" tab. The tab that shows the results of this scenario includes a TreeView

which can be expanded, and a second pane which is filled with additional information upon clicking the ID of a ligand, receptor, or result (Figure 8.24).

9.5 Conclusion

The flow of required process steps for each usability test (Figures 9.1, 9.2, and 9.3) can be used to compare completing a scenario "with" and "without" the implementation. One observation is that the usability tests "with" the implementation contain the same number or less manual steps than the usability tests "without" the implementation. This is because most of the steps are automated. In particular, there is no need to locate files on the file system, or manually read through web pages.

The decrease of manual steps is a major benefit in terms of usability. Another benefit is the decrease of complexity of the manual steps. This can be observed in each usability test "with" the implementation. Generally, the user needs to prepare the docking or VS in Raccoon2, enter the required user-provided inputs, and wait for the result of the scenario. When completing the scenario "without" the implementation, the user would still need to prepare the docking or VS, and then do several steps including some that may be considered complex for a biomedical scientist, such as writing and running a script.

Furthermore, the flowcharts of the scenarios "without" the implementation sometimes feature simple, but repetitive manual steps. For instance, uploading the structures of receptors and reading the results of the structural alignment multiple times. Although not very complex, these repetitive manual steps are very error-prone and time-consuming. Using the implementations does not have these problems because such steps have been automated.

Finally, completing the scenarios "with" or "without" the implementation requires a certain level of expertise. For instance, in order to interpret the results of the RaptorX website, the user should be aware of the meaning of the measures that are part of these results. Since the implementations require a user-provided threshold, the need for expertise or knowledge about the threshold value remains. However, a novice user could use the default threshold and still obtain meaningful results.

In summary, this section showed that running the scenarios "with" the implementations provides a degree of usability which is comparable to running the scenarios with currently available tools. In some cases an improvement in usability can be noticed. Therefore, the users of the implementations, the biomedical scientists, will also benefit from implementing this type of scenarios using the framework and methodology. This will produce usable

systems whose final results will be useful from a biomedical point of view. Unfortunately, it was not possible to have all five interviewees that participated in the primary research (Section 4.3) test the implementations.

Chapter 10

Conclusion

10.1 Summary of Thesis Achievements

Drugs have been discovered using "classical pharmacology" where the effects of a substance on an organism are determined first, and then the biological target is identified. Alternatively, in "reverse pharmacology", the biological target can be identified first, before discovering a suitable substance. The latter may include molecular docking simulations, which computationally estimate how and where two molecules would interact. This thesis shows how the development of computer systems that use docking results can be made easier for software developers while remaining useful for the biomedical scientists.

Firstly, a research gap was identified in the use of cloud computing for domain-specific desktop applications, such as large-scale docking applications. A generic concept for extending desktop applications with cloud computing capabilities was proposed, and tested for the large-scale docking tool Raccoon2 (Contribution 1, Chapter 3). Using this concept, popular desktop applications can be extended seamlessly and without major reengineering. The same desktop application, and the same familiar GUI can be used to leverage cloud computing resources. The biggest impact of the first contribution is that the generic concept can be used by software engineers to extend domain-specific desktop applications without major effort. As a result of this contribution, the tool Raccoon2 was extended, thus enabling Raccoon2 users to run docking simulations on various clouds. This increases the availability of the tool, particularly for users that do not have access to complex and expensive HPC clusters.

Secondly, the thesis explored the need to store molecular docking results, and the rationale behind it. Another research gap was identified: the lack of an openly available repository of docking results. With a repository, additional conclusions can be made,

based on the docking results that a scientist has created in the past, or based on previous results of other scientists. It would be useful for preventing the repetition of the same simulation, suggesting input files for a next docking simulation, or enabling novice users to learn how to conduct docking correctly by observing previous results. The proposed tool-independent conceptual framework (Contribution 2, Chapters 4, 5 and 6) is based on abstract descriptions of elements and interfaces. The abstract (diagrammatic, textual, and formal) description of a custom-made or existing tool can be used to determine whether it can be easily plugged into a scenario or whether an element implemented in one scenario can be reused in another scenario. Based on interviews with domain scientists, a set of scenarios that require a docking results repository were identified. The scenarios were used to define five generic element types of the framework, and the interfaces between them. The generic element types and interfaces of the framework have been verified through a literature review of existing systems.

Thirdly, this thesis showed how the framework can be used in a specifically defined methodology for developing software systems that use previous docking results (Contribution 3, Chapter 7). The methodology assumes regular communication among an interdisciplinary team and a planning and design part which includes the creation of a diagrammatic, textual, and formal description of the scenario. To avoid becoming overly cumbersome, it is based on the concept of agile methodologies. The main impact of the methodology is that it lists the three techniques (Section 7.3) where abstract descriptions, as specified by the framework, can be used to to determine: whether a new scenario fits the framework, whether an already implemented element can be reused in a new scenario, and whether a tool can be used as an element type. The methodical manner to develop this type of software systems will enable software developers to create less error-prone systems that reflect the aim of a scenario correctly.

Finally, to show how the framework and methodology can be used, Chapter 8 included reference implementations of three scenarios that have been identified through interviews with domain scientists. These scenarios emphasise different aspects of the framework, such as the ability to reuse already implemented elements or to use newly defined ones. The implementations provided examples of how the techniques of the methodology can benefit software developers. Chapter 9 shows that the use of the framework and methodology produces implementations that are usable by biomedical scientists and not overly complicated. This is shown through usability tests that illustrate how completing a scenario "with" the implementation has at least the same level of usability as completing a scenario using only currently available tools. In fact, the implementations require less manual steps and include more automatised processes. Generally, the required steps are less complex and require the same or lower level of expertise.

10.2 Future Work

One of the most important next steps would be to automate the comparison of abstract descriptions when deciding whether an already implemented element can be reused. In order to achieve this there are three types of comparisons of the three types of abstract descriptions: diagrammatic, textual, and formal. Before implementing automatic comparison tools, a database of abstract descriptions would need to be implemented. It would include optimised database or a novel data structure for storing the text (list of interfaces), diagrams (each diagram would need to be encoded in a format such as XML), and Z notation.

Once the database is created it can be filled with the abstract descriptions of all already implemented elements, including the ones part of the prototype implementations provided in this thesis. Then a method, or possibly three separate methods, for comparing the abstract descriptions would need to be created. The main concept should be to compare the types of interfaces and the core computation of the required element. An objective measure of what is similar enough to enable reuse would need to be devised for both the types of interfaces and for the core computation. The textual description can be mainly used for comparison of the interfaces. The formal description can provide more detail about the core computation. The diagrammatic description provides an overall picture of where the required element is in the scenario.

A powerful tool provided by Z notation and similar formal methods, which has not been explored in this thesis, is the mathematical proof - proof of correctness of the design and proof that the implementation behaves according the specification. Using Z, one can prove that the implementation of a system is consistent with the specification through the use of concepts such as *refinement*, where one specification is a low-level refinement of a more abstract specification. The main reason that a mathematical proof was not explored in this thesis is the fact that the formal description of the framework is abstract by definition. As future work, it is worth exploring if the formal description of the framework should be more detailed. It may be possible to remove non-determinism or uncertainty from some points of the framework's formal description. Most refinement procedures deal with a formal description on a much lower level than the formal description of the framework [168].

The formal description of each implemented scenario is more specific. However, it is mainly concerned with describing the specific elements for that scenario and the interfaces between them. As part of this thesis, it was shown that each description of an element is derived from the appropriate generic element type of the framework. A mathematical proof, which could involve a refinement of the formal description, would provide further guarantees that the element will be implemented as the specification has intended.

Furthermore, the implementations themselves can be improved in future research exercises. The aim of this thesis was not to create the ideal implementation for any scenario, but rather to provide a reference prototype implementation that shows the usefulness of the framework and methodology. There are several areas where the implementations can be improved. Namely, some of the key ones are:

- Defining the most advantageous data structure for storing the molecular information about ligands and receptors. In Chapter 2, the use of MSML [143], a part of the MoSGrid [145] system, was mentioned. The implementations presented in this thesis used a simpler JSON-based method as part of the MongoDB database. Each line of the respective .pdbqt file represents an element in a JSON array that describes the structure of the ligand or receptor. This was a pragmatic choice for the reference implementations. A future effort to compare and assess existing (e.g. MSML), or create an alternative new data structure with specific benefits, could be considered.
- Creating a user-independent manner to assess docking, receptor structural alignment, or ligand similarity tool. The current implementations use a method based on a user-provided threshold which is compared to a result value produced by the tool. This is not an uncommon method in VS pipelines [212]. However, this method can be improved. If annotations by a human expert are added to set of results, this can form the beginning of a training set for a method based on machine learning. For instance, a supervised learning method can be used to determine if a docking result is "good" or not. Similarly, annotating the results of the structural alignment or ligand similarity tools can be the first step towards using machine learning to assess what should be considered a pair of "similar" receptors or ligands.

Appendix A

Analysis of interviews with interviewees A-D

What scientists would use the system for?

1. Search all docking runs based on a protein, a ligand, or both

   (i) C said that for the scientist that is doing research on a certain protein, it would be useful to search for that protein (and species) and see everything that has already been docked to it.

   (ii) It would be useful to search for all the proteins that a certain ligand has been docked to.

   (iii) C added that it would be very useful to be able to search for all docking results of a certain protein-ligand pair and see if a particular combination has been docked before.

   (iv) Also, if a certain protein of the exact same species hasn't been docked, it would be very useful to see results for the similar protein from another species.

2. Redo simulation on the cloud

   (i) C thought it would be more useful to redo the simulation on the cloud compared to on your local PC, but wasn not completely convinced.

   (ii) A thought it is important to have this functionality. However, as a reason they stated "collaboration" which is not very related. Also, A said that the docking processes are stochastic. This is very important as every docking result is unique. It is entirely possible that something different comes out because it is more or less a statistical method.

(iii) D thought that if all the tools needed were pre-installed on a VM it would be more useful but they reckons it will be difficult.

3. Redo simulation on PC

(i) This sounded useful to C, because one may want to download the information about a certain docking, then check that it really works, and perhaps use it for something else.

(ii) A thought it would be quite informative and quite important.

(iii) B thought this is absolutely important, mainly because usually, you get good results and they translate to a published paper. But, if you expect some results to translate to a paper, and they don't - you want to know why. This is when you would like to rerun (redo) the simulation to check what the problem is.

(iv) D thinks automatically redoing the simulations on one's own PC would be useful but very difficult as there are many tools that need configuration and installing them is not simple

4. Just download input output and intermediate files

(i) According to C, viewing and downloading input files is more important than result files.

(ii) A rated them in this order: download intermediate files 7-8, download input files 9, and download the results 9-10. If you see some results, then see a paper where they have been published, it would be very nice to see the trajectory which led to the results.

(iii) In order of importance, B would order them in this way: input, output and then intermediate.

5. Contact scientist who did the simulation

(i) C gave 6/10 for contacting the people who performed the simulation.

(ii) A thought it may be useful sometimes but gave it only 3-4.

(iii) B said that if there seems to be an error in someone else's simulation results, contacting the owner will be useful if it is an ongoing collaboration. However if it is an old project and a died out collaboration, then it is not worth bothering.

(iv) B also added that it would be useful to have an annotation functionality where you can add a note and say that you have checked these results and got something different, so that later people will see this.

(v) Also, B believes that it would be very useful, in fact improve research practice, if it can be visible that you have done some simulations and improved

something and then get contacted by the original researcher. This would result in a collaboration. Rather than contacting someone who has done something directly, if you can show off your work on the same topic you could get their attention.

(vi) D gave 10/10 to contacting people and organisations that performed the simulations.

6. Compare your results with the same simulation with a different tool

   (i) C thinks that it would be very useful to include results from different docking tools because then you could compare results that someone else has got with some tool different than your favourite one and see if the results are similar. However, this would be very difficult as different tools use different input files and produce different result files, so there would need to be a way to view and analyse all these different types of files.

   (ii) A agreed that it is important to look at some information about the software tool that has been used and rated it 7-8.

7. Compare results of past simulations

   (i) C rated comparing past simulations (either yours or someone else's) with 7.

   (ii) A thought it would be useful and rated it with 8-9.

   (iii) D gave 9/10 to comparing results of past simulations.

What scientists think that this system should store?

1. Intermediate files

   (i) When discussing what to store in a database, A proposes to store the configuration files for MD runs and docking runs among others.

   (ii) A said that it is important to store the intermediate files if for instance your simulation gets interrupted (e.g. by electricity failure) and you want to restart it. You would use the intermediate files to know where to continue from.

   (iii) B referred to all the files that are created between tools, or between steps of the entire simulation process. Since there are usually many steps that go between the original input and the final output files, there are many files that get created. Sometimes these intermediate files can be disposable (e.g. if the only difference is changing the format). B stores all intermediate files in a non-backup server, meaning they may not be retrievable. But everything that is in the "working directory" should be stored including all the scripts that have been

used. The intermediate files aren't always useful for analysis, the final output files are the most useful ones containing the information you usually need. Still, the intermediate files are useful for teaching new students or if you revisit the same project at a later stage. Asked if it is possible to reproduce the results without the intermediate files, B explained how it is possible but it would take much longer and having the entire working directory with all the files will help students estimate how long will each step take and which is a more important step.

2. Log files
   (i) B would like to store logs of the intermediate steps, if not all the files themselves, having logs of the runs will be very useful.
   (ii) B believes that it will be very useful to have a system for keeping logs - a system of log files that make it easier to know what is the log of the entire run, what is the log of an individual step.
3. Peer-reviewed paper
   (i) Just like protein sequences and 3D crystal structures are published on PDB and Uniprot, associated with a paper that is then pending review, it would be a good idea to include a link to a published paper along with the docking results.
   (ii) A believes that storing a peer-reviewed paper together with the results would be useful.
      (a) a. Only once published
   (iii) C thinks people would share the record on their simulations only after they have published, otherwise they will fear the results would be used by someone else (get stolen).
4. Result files
   (i) A thinks the results should be stored at the university and every result should be stored. Actually after a slight correction - a single result from every trial should be stored.
   (ii) Referring to MD result files, A notes that these result files are huge, often gigabytes and there is a problem to store them and transfer them through the Internet.
   (iii) B mentions that storing input, intermediate and result files would be useful.
5. Structure of Molecules (Input files)
   (i) A thinks that the structure of the molecules itself needs to be stored. Apart from the structure and the resulting energy values the rest can be omitted.

(ii) Storing the input files, the receptor structure in particular should be stored in a database according to A, as there is a problem of finding the correct location and the correct folder which has been used to start previous simulation runs.

(iii) A thinks that the original files, that were downloaded for instance from the PDB, should be stored and not just the link because the files may change on the PDB

(iv) B explained a "4 copy rule" apparently used in photography, where the original file is left untouched and then altered in discrete stages until the final 4^{th} stage produces the final image (in this case the final result file). In other words, in B's practise the original input files are stored along with the parameter and configuration files. Basic test simulations are run using these to make sure they are OK and to optimise them if necessary. Once confirmed that they work properly, a new copy is made in a new folder with a date reference. All files from all steps are stored.

(v) B noted that it is absolutely important to have the original source saved, including the ID.

6. Who and when ran the simulation

(i) B would want a system that stores the name of the person that ran the simulation and the date and time, as it would help verify if the simulations have actually been run.

Appendix B

Formal Description of Framework for Systems that Use Docking Results

This appendix contains the formal description of element types and interfaces of the framework in Z notation. The choice of variable names should act as an additional explanation of the specification.

$[CHAR, DATE, ADDITIONAL\_TOOL\_RESULT, DATA\_SOURCE\_INPUT, DECISION]$

$LIGAND == \text{seq}\, CHAR$

$RECEPTOR == \text{seq}\, CHAR$

$CONFIG == \text{seq}\, CHAR$

$RESULT == \text{seq}\, CHAR$

$USER\_INPUT == \text{seq}\, CHAR$

$DATA\_SOURCE\_INFO == \text{seq}\, CHAR$

$LIGANDS == \mathbb{P}\, LIGAND$

$RECEPTORS == \mathbb{P}\, RECEPTOR$

$RESULTS == \mathbb{P}\, RESULT$

$PREVIOUS\_RESULT == (LIGAND \times RECEPTOR \times CONFIG \times DATE) \mapsto RESULT$

$$dockingWithoutConfig : (LIGAND \times RECEPTOR) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists res : RESULT \bullet dockingWithoutConfig(l, r) = res$$

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking

$$ligand? : LIGAND$$
$$receptor? : RECEPTOR$$
$$config? : CONFIG$$
$$result! : RESULT$$

$$config? = \varnothing \land result! = dockingWithoutConfig(ligand?, receptor?) \lor$$
$$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$$

MolecularDockingEnvironment

$$ligands? : LIGANDS$$
$$receptors? : RECEPTORS$$
$$config? : CONFIG$$
$$results! : RESULTS$$
$$date! : DATE$$

$$\exists ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking$$

ViewMolecularDockingResults

$$\Xi MolecularDockingEnvironment$$

$$results! \neq \varnothing$$

MolecularDockingResultsRepository

$$repository : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$$
$$decisionRepository : \{PREVIOUS\_RESULT\} \leftrightarrow DECISION$$

$$repository \neq \varnothing$$

InsertUpdateMolecularDockingResultsRepository1

$\Delta MolecularDockingResultsRepository$

$l? : LIGAND$

$r? : RECEPTOR$

$c? : CONFIG$

$res? : RESULT$

$d? : DATE$

$repository' = repository \oplus \{(l?, r?, c?, d?) \mapsto res?\}$

InsertUpdateMolecularDockingResultsRepositoryMany

$\Delta MolecularDockingResultsRepository$

$dockingResults? : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$

$l : LIGAND$

$r : RECEPTOR$

$c : CONFIG$

$d : DATE$

$\{(l, r, c, d)\} = \mathrm{dom}(dockingResults?)$

$\forall res : dockingResults? \mid \{(l, r, c, d)\} \bullet repository' = repository \oplus \{(l, r, c, d) \mapsto res\}$

InsertUpdateDecisionRepository

$\Delta MolecularDockingResultsRepository$

$previousDockingResults? : \{PREVIOUS\_RESULT\}$

$decision? : DECISION$

$decisionRepository' = decisionRepository \oplus \{previousDockingResults? \mapsto decision?\}$

SelectMolecularDockingResults

$\Xi MolecularDockingResultsRepository$
$whereL? : LIGAND$
$whereR? : RECEPTOR$
$whereC? : CONFIG$
$whereD? : DATE$
$whereRes? : RESULT$
$selectResults! : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$
$lig : LIGAND$
$rec : RECEPTOR$
$con : CONFIG$
$dat : DATE$

$selectResults! = \{(whereL?, whereR?, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, con, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, con, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, con, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, con, dat)\} \lhd repository \lor$
$selectResults! = repository \rhd \{whereRes?\}$

$additionalTool\_PR : \{PREVIOUS\_RESULT\} \leftrightarrow$
$\quad ADDITIONAL\_TOOL\_RESULT$

$\exists\, pr : \{PREVIOUS\_RESULT\} \bullet$
$\quad \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_PR(pr) = atr$

$$
\begin{array}{|l}
additionalTool\_DSI : \{DATA\_SOURCE\_INFO\} \leftrightarrow \\
\quad ADDITIONAL\_TOOL\_RESULT \\
\hline
\exists\, dsi : \{DATA\_SOURCE\_INFO\} \bullet \\
\quad \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_DSI(dsi) = atr
\end{array}
$$

$$
\begin{array}{|l}
additionalTool\_ATR : \{ADDITIONAL\_TOOL\_RESULT\} \leftrightarrow \\
\quad ADDITIONAL\_TOOL\_RESULT \\
\hline
\exists\, another\_atr : \{ADDITIONAL\_TOOL\_RESULT\} \bullet \\
\quad \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_ATR(another\_atr) = atr
\end{array}
$$

$$
\begin{array}{|l}
additionalTool\_DSI\_PR : (\{DATA\_SOURCE\_INFO\} \times \{PREVIOUS\_RESULT\}) \leftrightarrow \\
\quad ADDITIONAL\_TOOL\_RESULT \\
\hline
\exists\, dsi : \{DATA\_SOURCE\_INFO\};\ pr : \{PREVIOUS\_RESULT\} \mid dsi \neq \varnothing \bullet \\
\quad \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_DSI\_PR(dsi, pr) = atr
\end{array}
$$

$$
\begin{array}{|l}
additionalTool\_UI\_PR : (USER\_INPUT \times \{PREVIOUS\_RESULT\}) \leftrightarrow \\
\quad ADDITIONAL\_TOOL\_RESULT \\
\hline
\exists\, ui : USER\_INPUT;\ pr : \{PREVIOUS\_RESULT\} \mid ui \neq \varnothing \bullet \\
\quad \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_UI\_PR(ui, pr) = atr
\end{array}
$$

$$
\begin{array}{|l}
additionalTool\_PR\_ATR : (\{PREVIOUS\_RESULT\} \times \\
\quad \{ADDITIONAL\_TOOL\_RESULT\}) \leftrightarrow ADDITIONAL\_TOOL\_RESULT \\
\hline
\exists\, pr : \{PREVIOUS\_RESULT\};\ another\_atr : \{ADDITIONAL\_TOOL\_RESULT\} \mid \\
\quad pr \notin \varnothing \land another\_atr \notin \varnothing \bullet \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet \\
\quad additionalTool\_PR\_ATR(pr, another\_atr) = atr
\end{array}
$$

$$
\begin{array}{|l}
additionalTool\_UI\_DSI : (USER\_INPUT \times \{DATA\_SOURCE\_INFO\}) \leftrightarrow \\
\quad ADDITIONAL\_TOOL\_RESULT \\
\hline
\exists\, ui : USER\_INPUT;\ dsi : \{DATA\_SOURCE\_INFO\} \mid ui \neq \varnothing \land dsi \neq \varnothing \bullet \\
\quad \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_UI\_DSI(ui, dsi) = atr
\end{array}
$$

$$additionalTool\_UI\_ATR : (USER\_INPUT \times \{ADDITIONAL\_TOOL\_RESULT\}) \leftrightarrow ADDITIONAL\_TOOL\_RESULT$$

$$\exists\, ui : USER\_INPUT;\ another\_atr : \{ADDITIONAL\_TOOL\_RESULT\} \mid ui \neq \varnothing \wedge another\_atr \neq \varnothing \bullet \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_UI\_ATR(ui, another\_atr) = atr$$

$$additionalTool\_DSI\_ATR : (\{DATA\_SOURCE\_INFO\} \times \{ADDITIONAL\_TOOL\_RESULT\}) \leftrightarrow ADDITIONAL\_TOOL\_RESULT$$

$$\exists\, dsi : \{DATA\_SOURCE\_INFO\};\ another\_atr : \{ADDITIONAL\_TOOL\_RESULT\} \mid dsi \neq \varnothing \wedge another\_atr \notin \varnothing \bullet \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_DSI\_ATR(dsi, another\_atr) = atr$$

$$additionalTool\_UI\_DSI\_PR : (USER\_INPUT \times \{DATA\_SOURCE\_INFO\} \times \{PREVIOUS\_RESULT\}) \leftrightarrow ADDITIONAL\_TOOL\_RESULT$$

$$\exists\, ui : USER\_INPUT;\ dsi : \{DATA\_SOURCE\_INFO\};\ pr : \{PREVIOUS\_RESULT\} \mid ui \neq \varnothing \wedge dsi \neq \varnothing \bullet \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_UI\_DSI\_PR(ui, dsi, pr) = atr$$

$$additionalTool\_PR\_UI\_ATR : (\{PREVIOUS\_RESULT\} \times USER\_INPUT \times \{ADDITIONAL\_TOOL\_RESULT\}) \leftrightarrow ADDITIONAL\_TOOL\_RESULT$$

$$\exists\, pr : \{PREVIOUS\_RESULT\};\ ui : USER\_INPUT;\ another\_atr : \{ADDITIONAL\_TOOL\_RESULT\} \mid pr \notin \varnothing \wedge ui \neq \varnothing \wedge another\_atr \notin \varnothing \bullet \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_PR\_UI\_ATR(pr, ui, another\_atr) = atr$$

$$additionalTool\_PR\_DSI\_ATR : (\{PREVIOUS\_RESULT\} \times \{DATA\_SOURCE\_INFO\} \times \{ADDITIONAL\_TOOL\_RESULT\}) \leftrightarrow ADDITIONAL\_TOOL\_RESULT$$

$$\exists\, pr : \{PREVIOUS\_RESULT\};\ dsi : \{DATA\_SOURCE\_INFO\};\ another\_atr : \{ADDITIONAL\_TOOL\_RESULT\} \mid pr \notin \varnothing \wedge dsi \neq \varnothing \wedge another\_atr \notin \varnothing \bullet \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_PR\_DSI\_ATR(pr, dsi, another\_atr) = atr$$

$$additionalTool\_UI\_DSI\_ATR : (USER\_INPUT \times \{DATA\_SOURCE\_INFO\} \times \{ADDITIONAL\_TOOL\_RESULT\}) \leftrightarrow ADDITIONAL\_TOOL\_RESULT$$

$$\exists\, ui : USER\_INPUT;\ dsi : \{DATA\_SOURCE\_INFO\};\ another\_atr : \{ADDITIONAL\_TOOL\_RESULT\} \mid ui \notin \varnothing \land dsi \neq \varnothing \land another\_atr \notin \varnothing \bullet \exists\, atr : ADDITIONAL\_TOOL\_RESULT \bullet additionalTool\_UI\_DSI\_ATR(ui, dsi, another\_atr) = atr$$

AdditionalTool

$userInput? : USER\_INPUT$

$dataSourceInfo? : \{DATA\_SOURCE\_INFO\}$

$previousDockingResults? : \{PREVIOUS\_RESULT\}$

$otherAdditionalToolsResults? : \{ADDITIONAL\_TOOL\_RESULT\}$

$additionalToolResult! : ADDITIONAL\_TOOL\_RESULT$

$additionalToolResult! = additionalTool\_PR(previousDockingResults?) \lor$

$additionalToolResult! = additionalTool\_DSI(dataSourceInfo?) \lor$

$additionalToolResult! = additionalTool\_ATR(otherAdditionalToolsResults?) \lor$

$additionalToolResult! = additionalTool\_DSI\_PR(dataSourceInfo?, previousDockingResults?) \lor$

$additionalToolResult! = additionalTool\_UI\_PR(userInput?, previousDockingResults?) \lor$

$additionalToolResult! = additionalTool\_PR\_ATR(previousDockingResults?, otherAdditionalToolsResults?) \lor$

$additionalToolResult! = additionalTool\_UI\_DSI(userInput?, dataSourceInfo?) \lor$

$additionalToolResult! = additionalTool\_UI\_ATR(userInput?, otherAdditionalToolsResults?) \lor$

$additionalToolResult! = additionalTool\_DSI\_ATR(dataSourceInfo?, otherAdditionalToolsResults?) \lor$

$additionalToolResult! = additionalTool\_UI\_DSI\_PR(userInput?, dataSourceInfo?, previousDockingResults?) \lor$

$additionalToolResult! = additionalTool\_PR\_UI\_ATR(previousDockingResults?, userInput?, otherAdditionalToolsResults?) \lor$

$additionalToolResult! = additionalTool\_PR\_DSI\_ATR(previousDockingResults?, dataSourceInfo?, otherAdditionalToolsResults?) \lor$

$additionalToolResult! = additionalTool\_UI\_DSI\_ATR(userInput?, dataSourceInfo?, otherAdditionalToolsResults?)$

ReadAnotherAdditionalToolResults

$\Delta AdditionalTool$

$oneOrMoreAdditionalToolsResults? : \{ADDITIONAL\_TOOL\_RESULT\}$

$otherAdditionalToolsResults?' =$
$\quad otherAdditionalToolsResults? \cup oneOrMoreAdditionalToolsResults?$

AdditionalDataSource

$repository : DATA\_SOURCE\_INPUT \leftrightarrow DATA\_SOURCE\_INFO$

$repository \neq \varnothing$

SelectAdditionalDataInfo

$\Xi AdditionalDataSource$

$dataSourceInput? : DATA\_SOURCE\_INPUT$

$dataSourceInfo! : \{DATA\_SOURCE\_INFO\}$

$selectedData : DATA\_SOURCE\_INPUT \leftrightarrow DATA\_SOURCE\_INFO$

$selectedData = \{(dataSourceInput?)\} \lhd repository$

$dataSourceInfo! = \mathrm{ran}(selectedData)$

$makeADecisionPreviousResults : \{PREVIOUS\_RESULT\} \leftrightarrow DECISION$

$\exists\, pr : \{PREVIOUS\_RESULT\} \bullet \exists\, d : DECISION \bullet$
$makeADecisionPreviousResults(pr) = d$

$makeADecisionUserInputPreviousResults : (USER\_INPUT \times$
$\{PREVIOUS\_RESULT\}) \leftrightarrow DECISION$

$\exists\, ui : USER\_INPUT;\ pr : \{PREVIOUS\_RESULT\} \mid ui \neq \varnothing \bullet \exists\, d : DECISION \bullet$
$makeADecisionUserInputPreviousResults(ui, pr) = d$

$$makeADecisionUserInputAdditionalToolPreviousResults : (USER\_INPUT \times \{ADDITIONAL\_TOOL\_RESULT\} \times \{PREVIOUS\_RESULT\}) \leftrightarrow DECISION$$

$$\exists\, ui : USER\_INPUT;\ atr : \{ADDITIONAL\_TOOL\_RESULT\};\ pr : \{PREVIOUS\_RESULT\} \mid ui \neq \varnothing \bullet \exists\, d : DECISION \bullet makeADecisionUserInputAdditionalToolPreviousResults(ui, atr, pr) = d$$

$$makeADecisionAdditionalToolPreviousResults : (ADDITIONAL\_TOOL\_RESULT \times \{PREVIOUS\_RESULT\}) \leftrightarrow DECISION$$

$$\exists\, atr : ADDITIONAL\_TOOL\_RESULT;\ pr : \{PREVIOUS\_RESULT\} \bullet \exists\, d : DECISION \bullet makeADecisionAdditionalToolPreviousResults(atr, pr) = d$$

$$makeADecisionUserInputAdditionalTool : (USER\_INPUT \times \{ADDITIONAL\_TOOL\_RESULT\}) \leftrightarrow DECISION$$

$$\exists\, ui : USER\_INPUT;\ atr : \{ADDITIONAL\_TOOL\_RESULT\} \bullet \exists\, d : DECISION \bullet makeADecisionUserInputAdditionalTool(ui, atr) = d$$

$$makeADecisionAdditionalTool : (\{ADDITIONAL\_TOOL\_RESULT\}) \leftrightarrow DECISION$$

$$\exists\, atr : \{ADDITIONAL\_TOOL\_RESULT\} \bullet \exists\, d : DECISION \bullet makeADecisionAdditionalTool(atr) = d$$

DecisionMaker

$userInput? : USER\_INPUT$

$additionalToolResult? : \{ADDITIONAL\_TOOL\_RESULT\}$

$previousDockingResults? : \{PREVIOUS\_RESULT\}$

$decision! : DECISION$

$decision! = makeADecisionUserInputAdditionalToolPreviousResults(userInput?,$
$additionalToolResult?, previousDockingResults?)$

$\lor$

$additionalToolResult? \in \varnothing \land decision! =$
$makeADecisionUserInputPreviousResults(userInput?, previousDockingResults?)$

$\lor$

$previousDockingResults? \in \varnothing \land decision! =$
$makeADecisionUserInputAdditionalTool(userInput?, additionalToolResult?)$

$\lor$

$userInput? = \varnothing \land ($
  $decision! = makeADecisionAdditionalToolPreviousResults($
    $additionalToolResult?, previousDockingResults?)$
  $\lor$
  $additionalToolResult? \in \varnothing \land decision! =$
    $makeADecisionPreviousResults(previousDockingResults?)$
  $\lor$
  $previousDockingResults? \in \varnothing \land decision! =$
    $makeADecisionAdditionalTool(additionalToolResult?))$

Framework

$mde : MolecularDockingEnvironment$

$mdrr : MolecularDockingResultsRepository$

$ats : \{AdditionalTool\}$

$adss : \{AdditionalDataSource\}$

$dm : DecisionMaker$

$mde \notin \varnothing \land mdrr \notin \varnothing \land dm \notin \varnothing$

$\forall ads : adss \bullet \exists at : ats \bullet SelectAdditionalDataInfo \neq \varnothing$

Appendix C

Formal Description of Scenario 1

section *Framework* **parents** *standard_toolkit*

$[CHAR, DATE, DECISION]$
$LIGAND == \mathrm{seq}\, CHAR$
$RECEPTOR == \mathrm{seq}\, CHAR$
$CONFIG == \mathrm{seq}\, CHAR$
$RESULT == \mathrm{seq}\, CHAR$
$LIGANDS == \mathbb{P}\, LIGAND$
$RECEPTORS == \mathbb{P}\, RECEPTOR$
$RESULTS == \mathbb{P}\, RESULT$
$DEEP\_ALIGN\_RESULT == \mathrm{seq}\, CHAR$
$YES\_NO ::= yes \mid no$
$USER\_INPUT == \mathrm{seq}\, CHAR$
$PREVIOUS\_RESULT == (LIGAND \times RECEPTOR \times CONFIG \times DATE) \mapsto RESULT$
$PREVIOUS\_RESULTS == \{PREVIOUS\_RESULT\}$

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking_AutoDockVina

$ligand? : LIGAND$

$receptor? : RECEPTOR$

$config? : CONFIG$

$result! : RESULT$

$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$

MolecularDockingEnvironment_Raccoon2

$ligands? : LIGANDS$

$receptors? : RECEPTORS$

$config? : CONFIG$

$results! : RESULTS$

$date! : DATE$

$\exists ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking\_AutoDockVina$

ViewMolecularDockingResults_Raccoon2

$\Xi MolecularDockingEnvironment\_Raccoon2$

$results! \neq \varnothing$

MolecularDockingResultsRepository_MongoDB

$repository : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$

$decisionRepository : \{PREVIOUS\_RESULT\} \leftrightarrow DECISION$

$repository \neq \varnothing$

InsertUpdateMolecularDockingResultsRepository1

$\Delta MolecularDockingResultsRepository\_MongoDB$

$l? : LIGAND$

$r? : RECEPTOR$

$c? : CONFIG$

$res? : RESULT$

$d? : DATE$

$repository' = repository \oplus \{(l?, r?, c?, d?) \mapsto res?\}$

InsertUpdateMolecularDockingResultsRepositoryMany

$\Delta MolecularDockingResultsRepository\_MongoDB$

$dockingResults? : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$

$l : LIGAND$

$r : RECEPTOR$

$c : CONFIG$

$d : DATE$

$\{(l, r, c, d)\} = \mathrm{dom}(dockingResults?)$

$\forall res : dockingResults? \ (\!|\ \{(l, r, c, d)\}\ |\!) \bullet repository' = repository \oplus \{(l, r, c, d) \mapsto res\}$

InsertUpdateDecisionRepository

$\Delta MolecularDockingResultsRepository\_MongoDB$

$previousDockingResults? : \{PREVIOUS\_RESULT\}$

$decision? : DECISION$

$decisionRepository' = decisionRepository \oplus \{previousDockingResults? \mapsto decision?\}$

SelectMolecularDockingResults

$\Xi MolecularDockingResultsRepository\_MongoDB$
$whereL? : LIGAND$
$whereR? : RECEPTOR$
$whereC? : CONFIG$
$whereD? : DATE$
$whereRes? : RESULT$
$selectResults! : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$
$lig : LIGAND$
$rec : RECEPTOR$
$con : CONFIG$
$dat : DATE$

$selectResults! = \{(whereL?, whereR?, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, con, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, con, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, con, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, con, dat)\} \lhd repository \lor$
$selectResults! = repository \rhd \{whereRes?\}$

$DeepAlignCore : (RECEPTOR \times RECEPTOR) \leftrightarrow DEEP\_ALIGN\_RESULT$

$\exists\, current\_receptor : RECEPTOR;\ previous\_receptor : RECEPTOR \bullet$
$\quad \exists\, dar : DEEP\_ALIGN\_RESULT \bullet$
$\quad DeepAlignCore(current\_receptor, previous\_receptor) = dar$

DeepAlign

$SelectMolecularDockingResults$

$previous\_receptor? : RECEPTOR$

$current\_receptor? : RECEPTOR$

$DeepAlignResult! : DEEP\_ALIGN\_RESULT$

$whereR? = previous\_receptor?$

$DeepAlignResult! = DeepAlignCore(current\_receptor?, previous\_receptor?)$

$goodDeepAlignResult : (DEEP\_ALIGN\_RESULT \times USER\_INPUT) \leftrightarrow YES\_NO$

$\forall dar : DEEP\_ALIGN\_RESULT;\ ui : USER\_INPUT \bullet \exists threshold : \mathbb{Z};\ DeepScore : \mathbb{Z} \bullet$

$DeepScore \geq threshold \land goodDeepAlignResult(dar, ui) = yes \lor$

$DeepScore < threshold \land goodDeepAlignResult(dar, ui) = no$

AssessDeepAlign

$DeepAlignResult? : DEEP\_ALIGN\_RESULT$

$userInput? : USER\_INPUT$

$r : RECEPTOR$

$assessed\_receptors! : RECEPTORS$

$r \in assessed\_receptors! \land goodDeepAlignResult(DeepAlignResult?, userInput?) = yes \lor$

$r \notin assessed\_receptors! \land goodDeepAlignResult(DeepAlignResult?, userInput?) = no$

$goodDocking : (RESULT \times USER\_INPUT) \leftrightarrow YES\_NO$

$\forall r : RESULT;\ ui : USER\_INPUT \bullet \exists threshold : \mathbb{Z};\ docking\_score : \mathbb{Z} \bullet$

$docking\_score \leq threshold \land goodDocking(r, ui) = yes \lor$

$docking\_score > threshold \land goodDocking(r, ui) = no$

AssessPreviousDocking

$SelectMolecularDockingResults$

$previous\_results? : PREVIOUS\_RESULTS$

$userInput? : USER\_INPUT$

$assessedPreviousResults! : PREVIOUS\_RESULTS$

$\exists\, previous\_result : previous\_results?;\ result : RESULT \bullet$

$\{result\} = \mathrm{ran}(previous\_result) \land$

$(previous\_result \in assessedPreviousResults! \land goodDocking(result, userInput?) = yes \lor$

$previous\_result \notin assessedPreviousResults! \land goodDocking(result, userInput?) = no)$

$makeADecisionAdditionalTool : (RECEPTORS \times PREVIOUS\_RESULTS) \leftrightarrow DECISION$

$\exists\, previous\_results : PREVIOUS\_RESULTS;\ receptors : RECEPTORS;$

$lig : LIGAND;\ d : DECISION;\ con : CONFIG;\ dat : DATE \bullet$

$\forall\, previous\_result : previous\_results;\ receptor : receptors \bullet$

$\{(lig, receptor, con, dat)\} = \mathrm{dom}(previous\_result)$

$\land$

$makeADecisionAdditionalTool(receptors, previous\_results) = d$

DecisionMaker_Custom

$assessed\_previous\_results? : PREVIOUS\_RESULTS$

$assessed\_receptors? : RECEPTORS$

$decision! : DECISION$

$decision! =$

$makeADecisionAdditionalTool(assessed\_receptors?, assessed\_previous\_results?)$

Appendix D

Formal Description of Scenario 2

section *Framework* **parents** *standard_toolkit*

$[CHAR, DATE, DECISION]$
$LIGAND == \mathrm{seq}\, CHAR$
$RECEPTOR == \mathrm{seq}\, CHAR$
$CONFIG == \mathrm{seq}\, CHAR$
$RESULT == \mathrm{seq}\, CHAR$
$LIGANDS == \mathbb{P}\, LIGAND$
$RECEPTORS == \mathbb{P}\, RECEPTOR$
$RESULTS == \mathbb{P}\, RESULT$
$DEEP\_ALIGN\_RESULT == \mathrm{seq}\, CHAR$
$LIGSIFT\_RESULT == \mathrm{seq}\, CHAR$
$YES\_NO ::= yes \mid no$
$USER\_INPUT == \mathrm{seq}\, CHAR$
$PREVIOUS\_RESULT == (LIGAND \times RECEPTOR \times CONFIG \times DATE) \mapsto RESULT$
$PREVIOUS\_RESULTS == \{PREVIOUS\_RESULT\}$

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking_AutoDockVina

$ligand? : LIGAND$
$receptor? : RECEPTOR$
$config? : CONFIG$
$result! : RESULT$

$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$

MolecularDockingEnvironment_Raccoon2

$ligands? : LIGANDS$
$receptors? : RECEPTORS$
$config? : CONFIG$
$results! : RESULTS$
$date! : DATE$

$\exists ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking\_AutoDockVina$

ViewMolecularDockingResults

$\Xi MolecularDockingEnvironment$

$results! \neq \varnothing$

MolecularDockingResultsRepository

$repository : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$
$decisionRepository : \{PREVIOUS\_RESULT\} \leftrightarrow DECISION$

$repository \neq \varnothing$

InsertUpdateMolecularDockingResultsRepository1

$\Delta MolecularDockingResultsRepository$

$l? : LIGAND$

$r? : RECEPTOR$

$c? : CONFIG$

$res? : RESULT$

$d? : DATE$

$repository' = repository \oplus \{(l?, r?, c?, d?) \mapsto res?\}$

InsertUpdateMolecularDockingResultsRepositoryMany

$\Delta MolecularDockingResultsRepository$

$dockingResults? : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$

$l : LIGAND$

$r : RECEPTOR$

$c : CONFIG$

$d : DATE$

$\{(l, r, c, a, d)\} = \mathrm{dom}(dockingResults?)$

$\forall res : dockingResults? \mid \{(l, r, c, a, d)\} \bullet repository' = repository \oplus \{(l, r, c, a, d) \mapsto res\}$

InsertUpdateDecisionRepository

$\Delta MolecularDockingResultsRepository$

$previousDockingResults? : \{PREVIOUS\_RESULT\}$

$decision? : DECISION$

$decisionRepository' = decisionRepository \oplus \{previousDockingResults? \mapsto decision?\}$

SelectMolecularDockingResults

$\Xi MolecularDockingResultsRepository$

$whereL? : LIGAND$

$whereR? : RECEPTOR$

$whereC? : CONFIG$

$whereD? : DATE$

$whereRes? : RESULT$

$selectResults! : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$

$lig : LIGAND$

$rec : RECEPTOR$

$con : CONFIG$

$dat : DATE$

$selectResults! = \{(whereL?, whereR?, whereC?, whereD?)\} \lhd repository \lor$

$selectResults! = \{(whereL?, whereR?, whereC?, dat)\} \lhd repository \lor$

$selectResults! = \{(whereL?, whereR?, con, whereD?)\} \lhd repository \lor$

$selectResults! = \{(whereL?, whereR?, con, dat)\} \lhd repository \lor$

$selectResults! = \{(whereL?, rec, whereC?, whereD?)\} \lhd repository \lor$

$selectResults! = \{(whereL?, rec, whereC?, dat)\} \lhd repository \lor$

$selectResults! = \{(whereL?, rec, con, whereD?)\} \lhd repository \lor$

$selectResults! = \{(whereL?, rec, con, dat)\} \lhd repository \lor$

$selectResults! = \{(lig, whereR?, whereC?, whereD?)\} \lhd repository \lor$

$selectResults! = \{(lig, whereR?, whereC?, dat)\} \lhd repository \lor$

$selectResults! = \{(lig, whereR?, con, whereD?)\} \lhd repository \lor$

$selectResults! = \{(lig, whereR?, con, dat)\} \lhd repository \lor$

$selectResults! = \{(lig, rec, whereC?, whereD?)\} \lhd repository \lor$

$selectResults! = \{(lig, rec, whereC?, dat)\} \lhd repository \lor$

$selectResults! = \{(lig, rec, con, whereD?)\} \lhd repository \lor$

$selectResults! = \{(lig, rec, con, dat)\} \lhd repository \lor$

$selectResults! = repository \rhd \{whereRes?\}$

$goodDocking : (RESULT \times USER\_INPUT) \leftrightarrow YES\_NO$

$\forall r : RESULT;\ ui : USER\_INPUT \bullet \exists threshold : \mathbb{Z};\ docking\_score : \mathbb{Z} \bullet$

$docking\_score \leq threshold \land goodDocking(r, ui) = yes$

$\lor$

$docking\_score > threshold \land goodDocking(r, ui) = no$

AssessPreviousDocking

$SelectMolecularDockingResults$
$previous\_results? : PREVIOUS\_RESULTS$
$userInput? : USER\_INPUT$
$assessedPreviousResults! : PREVIOUS\_RESULTS$

$\exists\, previous\_result : previous\_results?;\ result : RESULT \bullet$
$\{result\} = \mathrm{ran}(previous\_result) \land$
$(previous\_result \in assessedPreviousResults! \land goodDocking(result, userInput?) = yes$
$\lor$
$previous\_result \notin assessedPreviousResults! \land goodDocking(result, userInput?) = no)$

$checkPubChem : (LIGAND \times USER\_INPUT) \leftrightarrow YES\_NO$

$\forall\, l : LIGAND;\ ui : USER\_INPUT \bullet \exists\, ligand\_property : ui \bullet$
$checkPubChem(l, ui) = yes$
$\lor$
$checkPubChem(l, ui) = no$

PubChem

$SelectMolecularDockingResults$
$previous\_results? : PREVIOUS\_RESULTS$
$ui? : USER\_INPUT$
$filtered\_results! : PREVIOUS\_RESULTS$
$rec : RECEPTOR$
$con : CONFIG$
$dat : DATE$
$previous\_result : PREVIOUS\_RESULT$

$\forall\, lig : LIGAND \bullet whereL? = lig;\ previous\_result \in previous\_results?;$
$\{(lig, rec, con, dat)\} = \mathrm{dom}(previous\_result);$
$selectResults! \in filtered\_results! \land checkPubChem(lig, ui?) = yes$
$\lor$
$selectResults! \notin filtered\_results! \land checkPubChem(lig, ui?) = no$

$$
\begin{array}{|l}
makeADecisionPreviousResults : (PREVIOUS\_RESULTS \times PREVIOUS\_RESULTS) \\
\quad \leftrightarrow DECISION \\
\hline
\exists\, previous\_results\_filtered\_ligands : PREVIOUS\_RESULTS; \\
\quad previous\_results\_assessed\_docking : PREVIOUS\_RESULTS; \\
\quad d : DECISION; \bullet \\
\quad\quad \forall\, previous\_result\_filtered\_ligands : previous\_results\_filtered\_ligands; \\
\quad\quad\quad previous\_result\_assessed\_docking : previous\_results\_assessed\_docking \bullet \\
\quad\quad\quad\quad previous\_result\_filtered\_ligands = previous\_result\_assessed\_docking \land \\
\quad\quad\quad\quad makeADecisionPreviousResults(\\
\quad\quad\quad\quad\quad previous\_results\_filtered\_ligands, previous\_results\_assessed\_docking) = d
\end{array}
$$

DecisionMaker_Custom

$$
\begin{array}{|l}
assessed\_previous\_results? : PREVIOUS\_RESULTS \\
filtered\_previous\_results? : PREVIOUS\_RESULTS \\
decision! : DECISION \\
\hline
decision! = \\
makeADecisionPreviousResults(assessed\_previous\_results?, filtered\_previous\_results?) \\
\hline
\end{array}
$$

Appendix E

Formal Description of Scenario 4

section *Framework* **parents** *standard_toolkit*

$[CHAR, DATE, DECISION]$
$LIGAND == \mathrm{seq}\, CHAR$
$RECEPTOR == \mathrm{seq}\, CHAR$
$CONFIG == \mathrm{seq}\, CHAR$
$RESULT == \mathrm{seq}\, CHAR$
$LIGANDS == \mathbb{P}\, LIGAND$
$RECEPTORS == \mathbb{P}\, RECEPTOR$
$RESULTS == \mathbb{P}\, RESULT$
$DEEP\_ALIGN\_RESULT == \mathrm{seq}\, CHAR$
$LIGSIFT\_RESULT == \mathrm{seq}\, CHAR$
$YES\_NO ::= yes \mid no$
$USER\_INPUT == \mathrm{seq}\, CHAR$
$PREVIOUS\_RESULT == (LIGAND \times RECEPTOR \times CONFIG \times DATE) \mapsto RESULT$
$PREVIOUS\_RESULTS == \{PREVIOUS\_RESULT\}$

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \rightarrowtail RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking_AutoDockVina

$ligand? : LIGAND$
$receptor? : RECEPTOR$
$config? : CONFIG$
$result! : RESULT$

$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$

MolecularDockingEnvironment_Raccoon2

$ligands? : LIGANDS$
$receptors? : RECEPTORS$
$config? : CONFIG$
$results! : RESULTS$
$date! : DATE$

$\exists ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking\_AutoDockVina$

ViewMolecularDockingResults

$\Xi MolecularDockingEnvironment$

$results! \neq \varnothing$

MolecularDockingResultsRepository

$repository : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$
$decisionRepository : \{PREVIOUS\_RESULT\} \leftrightarrow DECISION$

$repository \neq \varnothing$

InsertUpdateMolecularDockingResultsRepository1

$\Delta MolecularDockingResultsRepository$

$l? : LIGAND$

$r? : RECEPTOR$

$c? : CONFIG$

$res? : RESULT$

$d? : DATE$

$repository' = repository \oplus \{(l?, r?, c?, d?) \mapsto res?\}$

InsertUpdateMolecularDockingResultsRepositoryMany

$\Delta MolecularDockingResultsRepository$

$dockingResults? : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$

$l : LIGAND$

$r : RECEPTOR$

$c : CONFIG$

$d : DATE$

$\{(l, r, c, a, d)\} = \mathrm{dom}(dockingResults?)$

$\forall res : dockingResults? \lparr \{(l, r, c, d)\} \rparr \bullet repository' = repository \oplus \{(l, r, c, d) \mapsto res\}$

InsertUpdateDecisionRepository

$\Delta MolecularDockingResultsRepository$

$previousDockingResults? : \{PREVIOUS\_RESULT\}$

$decision? : DECISION$

$decisionRepository' = decisionRepository \oplus \{previousDockingResults? \mapsto decision?\}$

SelectMolecularDockingResults

$\Xi MolecularDockingResultsRepository$
$whereL? : LIGAND$
$whereR? : RECEPTOR$
$whereC? : CONFIG$
$whereD? : DATE$
$whereRes? : RESULT$
$selectResults! : (LIGAND \times RECEPTOR \times CONFIG \times DATE) \leftrightarrow RESULT$
$lig : LIGAND$
$rec : RECEPTOR$
$con : CONFIG$
$dat : DATE$

$selectResults! = \{(whereL?, whereR?, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, whereR?, con, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(whereL?, rec, con, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, whereR?, con, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, whereC?, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, whereC?, dat)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, con, whereD?)\} \lhd repository \lor$
$selectResults! = \{(lig, rec, con, dat)\} \lhd repository \lor$
$selectResults! = repository \rhd \{whereRes?\}$

$DeepAlignCore : (RECEPTOR \times RECEPTOR) \leftrightarrow DEEP\_ALIGN\_RESULT$

$\exists\, current\_receptor : RECEPTOR;\ previous\_receptor : RECEPTOR \bullet$
$\quad \exists\, dar : DEEP\_ALIGN\_RESULT \bullet$
$\quad DeepAlignCore(current\_receptor, previous\_receptor) = dar$

DeepAlign

$SelectMolecularDockingResults$
$previous\_receptor? : RECEPTOR$
$current\_receptor? : RECEPTOR$
$DeepAlignResult! : DEEP\_ALIGN\_RESULT$

$whereR? = previous\_receptor?$
$DeepAlignResult! = DeepAlignCore(current\_receptor?, previous\_receptor?)$

$goodDeepAlignResult : (DEEP\_ALIGN\_RESULT \times USER\_INPUT) \leftrightarrow YES\_NO$

$\forall dar : DEEP\_ALIGN\_RESULT;\ ui : USER\_INPUT \bullet \exists threshold : \mathbb{Z};\ DeepScore : \mathbb{Z} \bullet$
$DeepScore \geq threshold \land goodDeepAlignResult(dar, ui) = yes$
$\lor$
$DeepScore < threshold \land goodDeepAlignResult(dar, ui) = no$

AssessDeepAlign

$DeepAlignResult? : DEEP\_ALIGN\_RESULT$
$userInput? : USER\_INPUT$
$r : RECEPTOR$
$assessed\_receptors! : RECEPTORS$

$r \in assessed\_receptors! \land goodDeepAlignResult(DeepAlignResult?, userInput?) = yes$
$\lor$
$r \notin assessed\_receptors! \land goodDeepAlignResult(DeepAlignResult?, userInput?) = no$

$LIGSIFTCore : (LIGAND \times LIGAND) \leftrightarrow LIGSIFT\_RESULT$

$\exists current\_ligand : LIGAND;\ previous\_ligand : LIGAND \bullet$
$\exists lr : LIGSIFT\_RESULT \bullet$
$LIGSIFTCore(current\_ligand, previous\_ligand) = lr$

LIGSIFT

$SelectMolecularDockingResults$

$previous\_ligand? : LIGAND$

$current\_ligand? : LIGAND$

$LIGSIFTResult! : LIGSIFT\_RESULT$

$whereL? = previous\_ligand?$

$LIGSIFTResult! = LIGSIFTCore(current\_ligand?, previous\_ligand?)$

$goodLIGSIFTResult : (LIGSIFT\_RESULT \times USER\_INPUT) \leftrightarrow YES\_NO$

$\forall lr : LIGSIFT\_RESULT;\ ui : USER\_INPUT \bullet \exists threshold : \mathbb{Z};\ LIGSIFTScore : \mathbb{Z} \bullet$

$LIGSIFTScore \geq threshold \land goodLIGSIFTResult(lr, ui) = yes$

$\lor$

$LIGSIFTScore < threshold \land goodLIGSIFTResult(lr, ui) = no$

AssessLIGSIFT

$LIGSIFTResult? : LIGSIFT\_RESULT$

$userInput? : USER\_INPUT$

$l : LIGAND$

$assessed\_ligands! : LIGANDS$

$l \in assessed\_ligands! \land goodLIGSIFTResult(LIGSIFTResult?, userInput?) = yes$

$\lor$

$l \notin assessed\_ligands! \land goodLIGSIFTResult(LIGSIFTResult?, userInput?) = no$

$similarConfig : (CONFIG \times CONFIG \times USER\_INPUT) \leftrightarrow YES\_NO$

$\forall c1 : CONFIG;\ c2 : CONFIG;\ ui : USER\_INPUT \bullet \exists threshold : \mathbb{Z};$

$config\_similarity\_score : \mathbb{Z} \bullet$

$config\_similarity\_score > threshold \land similarConfig(c1, c2, ui) = yes$

$\lor$

$config\_similarity\_score \leq threshold \land similarConfig(c1, c2, ui) = no$

CompareMolecularDockingResults

$previous\_configs? : CONFIGS$
$current\_config? : CONFIG$
$userInput? : USER\_INPUT$
$assessed\_configs! : CONFIGS$

$\forall c : previous\_configs? \bullet$
$c = whereC? \land ($
$c \in assessed\_configs! \land similarConfig(current\_config?, c, userInput?) = yes$
$\lor\ c \notin assessed\_configs! \land similarConfig(current\_config?, c, userInput?) = no)$

$makeADecisionAdditionalTools : (LIGANDS \times RECEPTORS \times CONFIGS)$
$\leftrightarrow DECISION$

$\exists previous\_results : PREVIOUS\_RESULTS;\ receptors : RECEPTORS;$
$ligands : LIGANDS;\ configs : CONFIGS;$
$dat : DATE;\ d : DECISION \bullet$
$\forall receptor : receptors;\ ligand : ligands;\ config : configs;$
$previous\_result : previous\_results \bullet$
$\{(ligand, receptor, config, dat)\} = \mathrm{dom}(previous\_result)$
$\land$
$makeADecisionAdditionalTools(ligands, receptors, configs) = d$

DecisionMaker_Custom

$assessed\_receptors? : RECEPTORS$
$assessed\_ligands? : LIGANDS$
$assessed\_configs? : CONFIGS$
$decision! : DECISION$

$decision! = makeADecisionAdditionalTools($
$assessed\_ligands?, assessed\_receptors?, assessed\_configs?$
$)$

Appendix F

Amino Acids

The amino acids that feature in human proteins are often divided into these categories (the names can be abbreviated to a 3- or 1-letter version):

- Amino acids with charged side chains: side chains can make salt bridges (H-H) and gain an electrical charge:
  - Positively charged: Arginine-Arg-R, Histidine-His-H, Lysine-Lys-K.
  - Negatively charged: Aspartic acid-Asp-D, Glutamic acid-Glu-E.
- Amino acids with polar uncharged side chains (can participate in hydrogen bonds as proton donors or acceptors): Glutamine-Gln-Q, Asparagine-Asn-N, Serine-Ser-S, Threonine-Thr-T, Tyrosine-Tyr-Y, Tryptophan-Trp-W.
- Hydrophobic (normally buried inside the protein core): Alanine-Ala-A, Isoleucine-Ile-I, Leucine-Leu-L, Methionine-Met-M, Phenylalanine-Phe-F, Valine-Val-V.
- Special cases: Selenocysteine-Sec-U, Cysteine-Cys-C (sometimes categorised as polar), Glycine-Gly-G, Proline-Pro-P (sometimes categorised as hydrophobic).

Bibliography

[1] A. Fleming, "On the antibacterial action of cultures of a penicillium, with special reference to their use in the isolation of *B. influenzae*," *British Journal of Experimental Pathology*, vol. 10, no. 3, p. 226, 1929.

[2] J. W. Bennett and K.-T. Chung, "Alexander Fleming and the discovery of penicillin," in *Advances in Applied Microbiology*, vol. 49, pp. 163 – 184, Academic Press, 2001.

[3] T. Takenaka, "Classical vs reverse pharmacology in drug discovery," *BJU International*, vol. 88, no. s2, pp. 7–10, 2001.

[4] N. C. Cohen, "Medicine pipeline: Structure-based drug design and the discovery of Aliskiren (Tekturna®): Perseverance and creativity to overcome a R & D pipeline challenge," *Chemical Biology & Drug Design*, vol. 70, no. 6, pp. 557–565, 2007.

[5] E. Raviña, *The evolution of drug discovery: From traditional medicines to modern drugs*, ch. 6. Structure-Based Drug Discovery, pp. 405–441. John Wiley & Sons, 2011.

[6] Y. Bi, M. Might, H. Vankayalapati, and B. Kuberan, "Repurposing of Proton Pump Inhibitors as first identified small molecule inhibitors of endo-β-N-acetylglucosaminidase (ENGase) for the treatment of NGLY1 deficiency, a rare genetic disease," *Bioorganic & Medicinal Chemistry Letters*, vol. 27, no. 13, pp. 2962–2966, 2017.

[7] B. Srinivasan, H. Zhou, S. Mitra, and J. Skolnick, "Novel small molecule binders of human N-glycanase 1, a key player in the endoplasmic reticulum associated degradation pathway," *Bioorganic & Medicinal Chemistry*, vol. 24, no. 19, pp. 4750–4758, 2016.

[8] M. Might, "Hunting down my son's killer." Available at: `http://matt.might.net/articles/my-sons-killer/` [Accessed 6 Apr 2018].

[9] M. Might, "The illustrated guide to a Ph.D." Available at: `http://matt.might.net/articles/phd-school-in-pictures/` [Accessed 6 Apr 2018].

[10] F. Bowker, *Molecular docking and geographical information systems as tools to assess the potential impact of veterinary medicines on non-target organisms and the environment*. PhD thesis, University of Westminster, 2015.

[11] S. Gesing, S. Herres-Pawlis, G. Birkenheuer, A. Brinkmann, R. Grunzke, P. Kacsuk, O. Kohlbacher, M. Kozlovszky, J. Krüger, R. Müller-Pfefferkorn, *et al.*, "A science gateway getting ready for serving the international molecular simulation community," in *Proc. of the EGI Community Forum*, 2012.

[12] D. Temelkovski, T. Kiss, and G. Terstyanszky, "Molecular docking with Raccoon2 on clouds: Extending desktop applications with cloud computing," in *International Workshop of Science Gateways (IWSG) 2017*, p. 7, Jun 2017.

[13] D. Temelkovski, T. Kiss, and G. Terstyanszky, "A generic framework and methodology for implementing science gateways for analysing molecular docking results," in *International Workshop of Science Gateways (IWSG) 2018*, p. 7, Jun 2018.

[14] H. Berman, K. Henrick, and H. Nakamura, "Announcing the worldwide protein data bank," *Nature Structural and Molecular Biology*, vol. 10, no. 12, p. 980, 2003.

[15] D. Weininger, "SMILES, a chemical language and information system," *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, 1988.

[16] J. J. Irwin and B. K. Shoichet, "ZINC – a free database of commercially available compounds for virtual screening," *Journal of Chemical Information and Modeling*, vol. 45, no. 1, pp. 177–182, 2005.

[17] S. Kim, P. A. Thiessen, E. E. Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B. A. Shoemaker, *et al.*, "PubChem Substance and Compound databases," *Nucleic Acids Research*, vol. 44, no. D1, pp. D1202–D1213, 2015.

[18] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[19] A. Lesk, *Introduction to bioinformatics*. Oxford university press, 2014.

[20] H. Hasegawa and L. Holm, "Advances and pitfalls of protein structural alignment," *Current Opinion in Structural Biology*, vol. 19, no. 3, pp. 341–348, 2009.

[21] E. C. Meng, "Online structure alignment resources," Apr 2005. Available at: `http://www.rbvi.ucsf.edu/home/meng/grpmt/structalign.html` [Accessed 30 Mar 2018].

[22] E. Martz, W. Decatur, and M. Wiederstein, "Structural alignment tools," Oct 2016. Available at: `http://proteopedia.org/wiki/index.php/Structural_alignment_tools` [Accessed 30 Mar 2018].

[23] L. Holm and C. Sander, "Protein structure comparison by alignment of distance matrices," *Journal of Molecular Biology*, vol. 233, no. 1, pp. 123–138, 1993.

[24] I. N. Shindyalov and P. E. Bourne, "Protein structure alignment by incremental combinatorial extension (CE) of the optimal path," *Protein Engineering*, vol. 11, no. 9, pp. 739–747, 1998.

[25] Y. Zhang and J. Skolnick, "TM-align: A protein structure alignment algorithm based on the TM-score," *Nucleic Acids Research*, vol. 33, no. 7, pp. 2302–2309, 2005.

[26] A. S. Konagurthu, J. C. Whisstock, P. J. Stuckey, and A. M. Lesk, "MUSTANG: A multiple structural alignment algorithm," *Proteins: Structure, Function, and Bioinformatics*, vol. 64, no. 3, pp. 559–574, 2006.

[27] A. R. Ortiz, C. E. Strauss, and O. Olmea, "MAMMOTH (matching molecular models obtained from theory): An automated method for model comparison," *Protein Science*, vol. 11, no. 11, pp. 2606–2621, 2002.

[28] M. Shatsky, R. Nussinov, and H. J. Wolfson, "A method for simultaneous alignment of multiple protein structures," *Proteins: Structure, Function, and Bioinformatics*, vol. 56, no. 1, pp. 143–156, 2004.

[29] J. Zhu and Z. Weng, "FAST: A novel protein structure alignment algorithm," *Proteins: Structure, Function, and Bioinformatics*, vol. 58, no. 3, pp. 618–627, 2005.

[30] M. Menke, B. Berger, and L. Cowen, "Matt: Local flexibility aids protein multiple structure alignment," *PLoS Computational Biology*, vol. 4, no. 1, p. e10, 2008.

[31] J. Konc and D. Janežič, "ProBiS algorithm for detection of structurally similar protein binding sites by local structural alignment," *Bioinformatics*, vol. 26, no. 9, pp. 1160–1168, 2010.

[32] A. Prlić, S. Bliven, P. W. Rose, W. F. Bluhm, C. Bizon, A. Godzik, and P. E. Bourne, "Pre-calculated protein structure alignments at the RCSB PDB website," *Bioinformatics*, vol. 26, no. 23, pp. 2983–2985, 2010.

[33] S. Wang, J. Ma, J. Peng, and J. Xu, "Protein structure alignment beyond spatial proximity," *Scientific Reports*, vol. 3, p. 1448, 2013.

[34] X. Yuan and C. Bystroff, "Non-sequential structure-based alignments reveal topology-independent core packing arrangements in proteins," *Bioinformatics*, vol. 21, no. 7, pp. 1010–1019, 2004.

[35] J. Shapiro and D. Brutlag, "FoldMiner: Structural motif discovery using an improved superposition algorithm," *Protein Science*, vol. 13, no. 1, pp. 278–294, 2004.

[36] A. D. Stivala, P. J. Stuckey, and A. I. Wirth, "Fast and accurate protein substructure searching with simulated annealing and GPUs," *BMC Bioinformatics*, vol. 11, no. 1, p. 446, 2010.

[37] M. Nguyen, K. P. Tan, and M. S. Madhusudhan, "CLICK – topology-independent comparison of biomolecular 3D structures," *Nucleic Acids Research*, vol. 39, no. suppl_2, pp. W24–W28, 2011.

[38] Y. Yang, J. Zhan, H. Zhao, and Y. Zhou, "A new size-independent score for pairwise protein structure alignment and its application to structure classification and nucleic-acid binding prediction," *Proteins: Structure, Function, and Bioinformatics*, vol. 80, no. 8, pp. 2080–2088, 2012.

[39] M. J. Sippl and M. Wiederstein, "Detection of spatial correlations in protein structures and molecular complexes," *Structure*, vol. 20, no. 4, pp. 718–728, 2012.

[40] S. Shi, Y. Zhong, I. Majumdar, S. Sri Krishna, and N. V. Grishin, "Searching for three-dimensional secondary structural patterns in proteins with ProSMoS," *Bioinformatics*, vol. 23, no. 11, pp. 1331–1338, 2007.

[41] S. Minami, K. Sawada, and G. Chikenji, "MICAN: A protein structure alignment algorithm that can handle multiple-chains, inverse alignments, c α only models, alternative alignments, and non-sequential alignments," *BMC Bioinformatics*, vol. 14, no. 1, p. 24, 2013.

[42] J. Ebert and D. Brutlag, "Development and validation of a consistency based multiple structure alignment algorithm," *Bioinformatics*, vol. 22, no. 9, pp. 1080–1087, 2006.

[43] H. Sun, A. Sacan, H. Ferhatosmanoglu, and Y. Wang, "Smolign: A spatial motifs-based protein multiple structural alignment method," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 1, pp. 249–261, 2012.

[44] A. Stivala, A. Wirth, and P. J. Stuckey, "Tableau-based protein substructure search using quadratic programming," *BMC bioinformatics*, vol. 10, no. 1, p. 153, 2009.

[45] P. Brown, W. Pullan, Y. Yang, and Y. Zhou, "Fast and accurate non-sequential protein structure alignment using a new asymmetric linear sum assignment heuristic," *Bioinformatics*, vol. 32, no. 3, pp. 370–377, 2015.

[46] F. Kaiser, A. Eisold, S. Bittrich, and D. Labudde, "Fit3D: A web application for highly accurate screening of spatial residue patterns in protein structure data," *Bioinformatics*, vol. 32, no. 5, pp. 792–794, 2015.

[47] D. Barthel, J. D. Hirst, J. Błażewicz, E. K. Burke, and N. Krasnogor, "ProCKSI: A decision support system for protein (structure) comparison, knowledge, similarity and information," *BMC Bioinformatics*, vol. 8, no. 1, p. 416, 2007.

[48] K. Mizuguchi, C. M. Deane, T. L. Blundell, and J. P. Overington, "HOMSTRAD: A database of protein structure alignments for homologous families," *Protein Science*, vol. 7, no. 11, pp. 2469–2471, 1998.

[49] A. Marchler-Bauer, S. Lu, J. B. Anderson, F. Chitsaz, M. K. Derbyshire, C. DeWeese-Scott, J. H. Fong, L. Y. Geer, R. C. Geer, N. R. Gonzales, *et al.*, "CDD: A conserved domain database for the functional annotation of proteins," *Nucleic Acids Research*, vol. 39, no. suppl_1, pp. D225–D229, 2010.

[50] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia, "SCOP: A structural classification of proteins database for the investigation of sequences and structures," *Journal of Molecular Biology*, vol. 247, no. 4, pp. 536–540, 1995.

[51] F. M. G. Pearl, C. Bennett, J. E. Bray, A. P. Harrison, N. Martin, A. Shepherd, I. Sillitoe, J. Thornton, and C. A. Orengo, "The CATH database: An extended protein family resource for structural and functional genomics," *Nucleic Acids Research*, vol. 31, no. 1, pp. 452–455, 2003.

[52] L. Holm and P. Rosenström, "Dali server: Conservation mapping in 3D," *Nucleic Acids Research*, vol. 38, no. suppl_2, pp. W545–W549, 2010.

[53] L. Holm and L. M. Laakso, "Dali server update," *Nucleic Acids Research*, vol. 44, no. W1, pp. W351–W355, 2016.

[54] C. Kim and B. Lee, "Accuracy of structure-based sequence alignment of automatic methods," *BMC Bioinformatics*, vol. 8, no. 1, p. 355, 2007.

[55] J. Havrilla and A. Saçan, "Meta-analysis of protein structural alignment," in *Bioinformatics and Biomedicine Workshops (BIBMW), 2012 IEEE International Conference on*, pp. 72–76, IEEE, 2012.

[56] R. Kolodny, P. Koehl, and M. Levitt, "Comprehensive evaluation of protein structure alignment methods: Scoring by geometric measures," *Journal of Molecular Biology*, vol. 346, no. 4, pp. 1173–1188, 2005.

[57] M. Källberg, H. Wang, S. Wang, J. Peng, Z. Wang, H. Lu, and J. Xu, "Template-based protein structure modeling using the RaptorX web server," *Nature Protocols*, vol. 7, no. 8, p. 1511, 2012.

[58] J. Ma and S. Wang, "Algorithms, applications, and challenges of protein structure alignment," in *Advances in Protein Chemistry and Structural Biology*, vol. 94, pp. 121–175, Elsevier, 2014.

[59] H. Eckert and J. Bajorath, "Molecular similarity analysis in virtual screening: Foundations, limitations and novel approaches," *Drug Discovery Today*, vol. 12, no. 5-6, pp. 225–233, 2007.

[60] T. Langer, "Pharmacophores in drug research," *Molecular Informatics*, vol. 29, pp. 470–475, Jul 2010.

[61] E. X. Esposito, A. J. Hopfinger, and J. D. Madura, "Methods for applying the quantitative structure–activity relationship paradigm," in *Chemoinformatics: Concepts, Methods, and Tools for Drug Discovery* (J. Bajorath, ed.), pp. 131–213, Totowa, NJ: Humana Press, 2004.

[62] B. Villoutreix, "vls3d.com: Ligand-based virtual screening," Feb 2018. Available at: `http://www.vls3d.com/links/chemoinformatics/virtual-screening/ligand-based-virtual-screening` [Accessed 30 Mar 2018].

[63] Z. Vincent and D. Antoine, "Click2Drug: Directory of in silico drug design tools," Sep 2017. Available at: `http://www.click2drug.org/index.html#Screening` [Accessed 30 Mar 2018].

[64] A. Roy and J. Skolnick, "LIGSIFT: An open-source tool for ligand structural alignment and virtual screening," *Bioinformatics*, vol. 31, no. 4, pp. 539–544, 2014.

[65] M. M. von Behren, S. Bietz, E. Nittinger, and M. Rarey, "mRAISE: An alternative algorithmic approach to ligand-based virtual screening," *Journal of Computer-Aided Molecular Design*, vol. 30, no. 8, pp. 583–594, 2016.

[66] M. M. Mysinger, M. Carchia, J. J. Irwin, and B. K. Shoichet, "Directory of useful decoys, enhanced (DUD-E): Better ligands and decoys for better benchmarking," *Journal of Medicinal Chemistry*, vol. 55, no. 14, pp. 6582–6594, 2012.

[67] J. Taminau, G. Thijs, and H. De Winter, "Pharao: Pharmacophore alignment and optimization," *Journal of Molecular Graphics and Modelling*, vol. 27, no. 2, pp. 161–169, 2008.

[68] J. A. Grant, M. Gallardo, and B. T. Pickup, "A fast method of molecular shape comparison: A simple application of a Gaussian description of molecular shape," *Journal of Computational Chemistry*, vol. 17, no. 14, pp. 1653–1666, 1996.

[69] M. J. Vainio, J. S. Puranen, and M. S. Johnson, "ShaEP: Molecular overlay based on shape and electrostatic potential," 2009.

[70] L. A. Vaz de Lima and A. S. Nascimento, "MolShaCS: A free and open source tool for ligand similarity identification based on Gaussian descriptors," *European Journal of Medicinal Chemistry*, vol. 59, pp. 296–303, 2013.

[71] G. Jones, P. Willett, R. C. Glen, A. R. Leach, and R. Taylor, "Development and validation of a genetic algorithm for flexible docking," *Journal of Molecular Biology*, vol. 267, no. 3, pp. 727–748, 1997.

[72] O. Trott and A. J. Olson, "AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," *Journal of Computational Chemistry*, pp. 455–461, 2009.

[73] A. Grosdidier, *EADock: Design of a new molecular docking algorithm and some of its applications*. PhD thesis, University of Lausanne, 2009.

[74] T. J. Ewing, S. Makino, A. G. Skillman, and I. D. Kuntz, "DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases," *Journal of Computer-Aided Molecular Design*, vol. 15, pp. 411–428, May 2001.

[75] J. Krüger, R. Grunzke, S. Herres-Pawlis, A. Hoffmann, L. de la Garza, O. Kohlbacher, W. E. Nagel, and S. Gesing, "Performance studies on distributed virtual screening," *BioMed Research Rnternational*, vol. 2014, 2014.

[76] UCSF DOCK, "The official UCSF DOCK web-site: Overview of DOCK." Available at `http://dock.compbio.ucsf.edu/Overview_of_DOCK/index.htm` [Accessed: 2 Apr 2018].

[77] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov, "Principles of docking: An overview of search algorithms and a guide to scoring functions," *Proteins: Structure, Function, and Genetics*, vol. 47, pp. 409–443, Jun 2002.

[78] S. F. Sousa, P. A. Fernandes, and M. J. Ramos, "Protein-ligand docking: Current status and future challenges," *Proteins: Structure, Function, and Bioinformatics*, vol. 65, pp. 15–26, Jul 2006. 00513.

[79] G. M. Morris, R. Huey, W. Lindstrom, M. F. Sanner, R. K. Belew, D. S. Goodsell, and A. J. Olson, "AutoDock4 and AutoDockTools4: Automated docking with

selective receptor flexibility," *Journal of Computational Chemistry*, vol. 30, no. 16, pp. 2785–2791, 2009.

[80] I. D. Kuntz, J. M. Blaney, S. J. Oatley, R. Langridge, and T. E. Ferrin, "A geometric approach to macromolecule-ligand interactions," *Journal of Molecular Biology*, vol. 161, no. 2, pp. 269–288, 1982.

[81] W. J. Allen, T. E. Balius, S. Mukherjee, S. R. Brozell, D. T. Moustakas, P. T. Lang, D. A. Case, I. D. Kuntz, and R. C. Rizzo, "DOCK 6: Impact of new features and current docking performance," *Journal of Computational Chemistry*, vol. 36, no. 15, pp. 1132–1156, 2015.

[82] R. G. Coleman, M. Carchia, T. Sterling, J. J. Irwin, and B. K. Shoichet, "Ligand pose and orientational sampling in molecular docking," *PLoS ONE*, vol. 8, pp. 1–19, Oct 2013.

[83] B. Kramer, M. Rarey, and T. Lengauer, "Evaluation of the FlexX incremental construction algorithm for protein–ligand docking," *Proteins: Structure, Function, and Bioinformatics*, vol. 37, no. 2, pp. 228–241, 1999.

[84] S. Cosconati, S. Forli, A. L. Perryman, R. Harris, D. S. Goodsell, and A. J. Olson, "Virtual screening with AutoDock: Theory and practice," *Expert Opinion on Drug Discovery*, vol. 5, no. 6, pp. 597–607, 2010.

[85] G. M. Morris, D. S. Goodsell, M. E. Pique, W. Lindstrom, R. Huey, S. Forli, W. E. Hart, S. Halliday, R. Belew, and A. J. Olson, "AutoDock Version 4.2: Updated for version 4.2.5," User Guide, Molecular Graphics Laboratory, the Scripps Research Institute, Nov 2012. Available at: `http://autodock.scripps.edu/faqs-help/manual/autodock-4-2-user-guide/AutoDock4.2_UserGuide.pdf` [Accessed 29 Mar 2018].

[86] G. M. Morris, "AD4_parameters.dat – AutoDock," Mar 2008. Available at: `http://autodock.scripps.edu/resources/parameters/AD4_parameters.dat/view` [Accessed 29 Mar 2018].

[87] S. Dallakyan, "MGLTools Website - Welcome - MGLTools," Apr 2010. Available at: `http://mgltools.scripps.edu/` [Accessed 29 Mar 2018].

[88] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, A. J. Olson, *et al.*, "Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function," *Journal of Computational Chemistry*, vol. 19, no. 14, pp. 1639–1662, 1998.

[89] R. Huey, G. M. Morris, A. J. Olson, and D. S. Goodsell, "A semiempirical free energy force field with charge-based desolvation," *Journal of Computational Chemistry*, vol. 28, no. 6, pp. 1145–1152, 2007.

[90] S. Moeller and G. M. Morris, "Ubuntu Manpage: autogrid - preparing protein and ligand for AutoDock analysis," Jul 2007. Available at: `http://manpages.ubuntu.com/manpages/xenial/en/man1/autogrid4.1.html` [Accessed 29 Mar 2018].

[91] R. Huey, "How to prepare a grid parameter file for AutoGrid 4 - AutoDock," May 2010. Available at: `http://autodock.scripps.edu/faqs-help/how-to/how-to-prepare-a-grid-parameter-files-for-autogrid4` [Accessed 29 Mar 2018].

[92] G. M. Morris, "Where do I set the AutoDock 4 force field parameters? - AutoDock," Aug 2007. Available at: `http://autodock.scripps.edu/faqs-help/faq/where-do-i-set-the-autodock-4-force-field-parameters` [Accessed 29 Mar 2018].

[93] M. W. Chang, C. Ayeni, S. Breuer, and B. E. Torbett, "Virtual screening for HIV protease inhibitors: A comparison of AutoDock 4 and Vina," *PLoS ONE*, vol. 5, no. 8, p. e11955, 2010.

[94] D. Seeliger and B. L. de Groot, "Ligand docking and binding site analysis with PyMOL and Autodock/Vina," *Journal of Computer-Aided Molecular Design*, vol. 24, no. 5, pp. 417–422, 2010.

[95] C. A. Lipinski, "Lead- and drug-like compounds: The rule-of-five revolution," *Drug Discovery Today: Technologies*, vol. 1, no. 4, pp. 337–341, 2004.

[96] M. Congreve, R. Carr, C. Murray, and H. Jhoti, "A "rule of three" for fragment-based lead discovery?," *Drug Discovery Today*, vol. 8, no. 19, pp. 876–877, 2003.

[97] D. F. Veber, S. R. Johnson, H.-Y. Cheng, B. R. Smith, K. W. Ward, and K. D. Kopple, "Molecular properties that influence the oral bioavailability of drug candidates," *Journal of Medicinal Chemistry*, vol. 45, no. 12, pp. 2615–2623, 2002.

[98] J. Biesiada, A. Porollo, and J. Meller, "On setting up and assessing docking simulations for virtual screening," in *Rational Drug Design*, pp. 1–16, Springer, 2012.

[99] S. Forli, "Raccoon - AutoDock VS Preparation Tool," Tech. Rep. User Manual, The Scripps Research Institute, Nov 2009. Available at `http://mgldev.scripps.edu/raccoon/Raccoon_v1.0_user_manual.pdf` [Accessed: 30 Mar 2018].

[100] S. Forli, R. Huey, M. E. Pique, M. F. Sanner, D. S. Goodsell, and A. J. Olson, "Computational protein-ligand docking and virtual drug screening with the AutoDock suite," *Nature Protocols*, vol. 11, no. 5, p. 905, 2016.

[101] S. Forli, "AutoDock | Raccoon2 - AutoDock," May 2016. Available at: `http://autodock.scripps.edu/resources/raccoon2` [Accessed 30 Mar 2018].

[102] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with NAMD," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[103] E. Chia, M. S. Shamsir, Z. A. Hussein, and S. Z. M. Hashim, "GridMACS portal: A grid web portal for molecular dynamics simulation using GROMACS," in *Mathematical/Analytical Modelling and Computer Simulation (AMS), 2010 Fourth Asia International Conference on*, pp. 507–512, IEEE, 2010.

[104] H. J. Berendsen, D. van der Spoel, and R. van Drunen, "GROMACS: A message-passing parallel molecular dynamics implementation," *Computer Physics Communications*, vol. 91, no. 1-3, pp. 43–56, 1995.

[105] E. Ford, *Building a Linux HPC Cluster with xCAT*. IBM Press, 2002.

[106] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer, "BEOWULF: A parallel workstation for scientific computation," in *Proceedings, International Conference on Parallel Processing*, vol. 95, pp. 11–14, 1995.

[107] M. A. Jette, A. B. Yoo, and M. Grondona, "SLURM: Simple linux utility for resource management," in *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pp. 44–60, Springer-Verlag, 2002.

[108] R. L. Henderson, "Job scheduling under the portable batch system," in *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 279–294, Springer, 1995.

[109] Altair, "PBS Professional overview." Available at: `https://www.pbsworks.com/PBSProduct.aspx?n=PBS-Professional&c=Overview-and-Capabilities` [Accessed 10 Aug 2018].

[110] W. Gentzsch, "Sun grid engine: Towards creating a compute power grid," in *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pp. 35–36, IEEE, 2001.

[111] "Grid engine – open grid scheduler." Available at: `http://gridscheduler.sourceforge.net/` [Accessed 10 Aug 2018].

[112] "TOP500 – The list: Develpoment over time." Available at: `https://www.top500.org/statistics/overtime/` [Accessed 10 Aug 2018].

[113] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, pp. 1–10, IEEE, 2008.

[114] P. Mell, T. Grance, *et al.*, "The NIST definition of cloud computing," tech. rep., Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

[115] T. Erl, R. Puttini, and Z. Mahmood, *Cloud computing: Concepts, technology & architecture*. Pearson Education, 2013.

[116] M. M. Jaghoori, A. J. Altena, B. Bleijlevens, S. Ramezani, J. L. Font, and S. D. Olabarriaga, "A multi-infrastructure gateway for virtual drug screening," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4478–4490, 2015.

[117] N. D. Prakhov, A. L. Chernorudskiy, and M. R. Gainullin, "VSDocker: A tool for parallel high-throughput virtual screening using AutoDock on Windows-based computer clusters," *Bioinformatics*, vol. 26, no. 10, pp. 1374–1375, 2010.

[118] X. Jiang, K. Kumar, X. Hu, A. Wallqvist, and J. Reifman, "DOVIS 2.0: An efficient and easy to use parallel virtual screening tool based on AutoDock 4.0," *Chemistry Central Journal*, vol. 2, no. 1, p. 18, 2008.

[119] I. Sánchez-Linares, H. Pérez-Sánchez, J. M. Cecilia, and J. M. García, "High-throughput parallel blind virtual screening using BINDSURF," *BMC Bioinformatics*, vol. 13, no. 14, p. S13, 2012.

[120] R. De Paris, F. A. Frantz, O. Norberto de Souza, and D. D. Ruiz, "wFReDoW: A cloud-based web environment to handle molecular docking simulations of a fully flexible receptor model," *BioMed Research International*, vol. 2013, 2013.

[121] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt, D. Ferguson, G. Seibel, and P. Kollman, "AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules," *Computer Physics Communications*, vol. 91, no. 1-3, pp. 1–41, 1995.

[122] S. R. Ellingson and J. Baudry, "High-throughput virtual molecular docking with AutoDockCloud," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 4, pp. 907–916, 2014.

[123] T. White, *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[124] T. Kiss, P. Borsody, G. Terstyanszky, S. Winter, P. Greenwell, S. McEldowney, and H. Heindl, "Large-scale virtual screening experiments on Windows Azure-based cloud resources," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 10, pp. 1760–1770, 2014.

[125] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[126] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, p. 316, 2017.

[127] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, *et al.*, "The Taverna workflow suite: Designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, no. W1, pp. W557–W561, 2013.

[128] P. Kacsuk, K. Karoczkai, G. Hermann, G. Sipos, and J. Kovacs, "WS-PGRADE: Supporting parameter sweep applications in workflows," in *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pp. 1–10, IEEE, 2008.

[129] S. J. Taylor, T. Kiss, G. Terstyanszky, P. Kacsuk, and N. Fantini, "Cloud computing for simulation in manufacturing and engineering: Introducing the CloudSME simulation platform," in *Proceedings of the 2014 Annual Simulation Symposium*, p. 12, Society for Computer Simulation International, 2014.

[130] M. Kozlovszky, K. Karóczkai, I. Márton, P. Kacsuk, and T. Gottdank, "DCI bridge: Executing WS-PGRADE workflows in distributed computing infrastructures," in *Science Gateways for Distributed Computing Infrastructures*, pp. 51–67, Springer, 2014.

[131] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, and I. Marton, "WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities," *Journal of Grid Computing*, vol. 10, no. 4, pp. 601–630, 2012.

[132] R. Sezov, *Liferay in action: The official guide to Liferay portal development.* Manning, 2011.

[133] Z. Farkas and P. Kacsuk, "P-GRADE portal: A generic workflow system to support user communities," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 454–465, 2011.

[134] A. Balasko, Z. Farkas, and P. Kacsuk, "Building science gateways by utilizing the generic WS-PGRADE/gUSE workflow system," *Computer Science*, vol. 14, no. 2, p. 307, 2013.

[135] P. Kacsuk, "P-GRADE portal family for grid infrastructures," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 3, pp. 235–245, 2011.

[136] T. Kiss, P. Kacsuk, R. Lovas, Á. Balaskó, A. Spinuso, M. Atkinson, D. D'Agostino, E. Danovaro, and M. Schiffers, "WS-PGRADE/gUSE in European projects," in *Science Gateways for Distributed Computing Infrastructures*, pp. 235–254, Springer, 2014.

[137] T. Gottdank, "Introduction to the WS-PGRADE/gUSE science gateway framework," in *Science Gateways for Distributed Computing Infrastructures* (P. Kacsuk, ed.), pp. 19–32, Springer, 2014.

[138] MTA SZTAKI LPDS, "Remote API configuration and specification," tech. rep., MTA SZTAKI LPDS, 2015. Available at `https://sourceforge.net/projects/guse/files/3.7.4/Documentation/RemoteAPI_Configuration.pdf` [Accessed: 1 Apr 2018].

[139] MTA SZTAKI LPDS, "Remote API tutorial: How to call WS-PGRADE workflows from remote clients through the http protocol?," tech. rep., MTA SZTAKI LPDS. Available at `https://www.sci-bus.eu/documents/94981/471552/Remote+API+tutorial` [Accessed: 1 Apr 2018].

[140] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, *et al.*, "PROV-DM: The PROV data model," tech. rep., World Wide Web Consortium, 2013.

[141] L. Moreau and P. Groth, *Provenance: An Introduction to PROV (Synthesis Lectures on the Semantic Web: Theory and Technology).* Morgan & Claypool Publishers, 2013.

[142] W3C, "RDF 1.1 Concepts and Abstract Syntax," W3C standard, Feb 2014. Available at `https://www.w3.org/TR/rdf11-concepts/` [Accessed: 1 Apr 2018].

[143] R. Grunzke, S. Breuers, S. Gesing, S. Herres-Pawlis, M. Kruse, D. Blunk, L. Garza, L. Packschies, P. Schäfer, C. Schärfe, *et al.*, "Standards-based metadata management for molecular simulations," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 10, pp. 1744–1759, 2014.

[144] M. McCandless, E. Hatcher, and O. Gospodnetic, *Lucene in action: Covers Apache Lucene 3.0*. Manning Publications Co., 2010.

[145] S. Gesing, R. Grunzke, J. Krüger, G. Birkenheuer, M. Wewior, P. Schäfer, B. Schuller, J. Schuster, S. Herres-Pawlis, S. Breuers, *et al.*, "A single sign-on infrastructure for science gateways on a use case for structural bioinformatics," *Journal of Grid Computing*, vol. 10, no. 4, pp. 769–790, 2012.

[146] J. Krüger, R. Grunzke, S. Gesing, S. Breuers, A. Brinkmann, L. de la Garza, O. Kohlbacher, M. Kruse, W. E. Nagel, L. Packschies, *et al.*, "The MoSGrid science gateway–a complete solution for molecular simulations," *Journal of Chemical Theory and Computation*, vol. 10, no. 6, pp. 2232–2245, 2014.

[147] O. Kohlbacher, "CADDSuite – a workflow-enabled suite of open-source tools for drug discovery," *Journal of Cheminformatics*, vol. 4, pp. 1–1, 2012.

[148] M. Rarey, B. Kramer, T. Lengauer, and G. Klebe, "A fast flexible docking method using an incremental construction algorithm," *Journal of Molecular Biology*, vol. 261, no. 3, pp. 470–489, 1996.

[149] G. Terstyanszky, T. Kukla, T. Kiss, P. Kacsuk, Á. Balaskó, and Z. Farkas, "Enabling scientific workflow sharing through coarse-grained interoperability," *Future Generation Computer Systems*, vol. 37, pp. 46–59, 2014.

[150] E. Spanoudakis, "Creating semantic applications using Taverna workflows," Master's thesis, University of Manchester, 2011.

[151] P. Missier, S. S. Sahoo, J. Zhao, C. Goble, and A. Sheth, "Janus: From workflows to semantic provenance and linked open data," in *International Provenance and Annotation Workshop*, pp. 129–141, Springer, 2010.

[152] C. A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, S. Bechhofer, M. Roos, P. Li, *et al.*, "myExperiment: A repository and social network for the sharing of bioinformatics workflows," *Nucleic Acids Research*, vol. 38, no. suppl_2, pp. W677–W682, 2010.

[153] S. Soiland-Reyes, "Apache Taverna Provenance (taverna-prov)," May 2016. Available at: `https://github.com/apache/incubator-taverna-engine` [Accessed 30 Mar 2018].

[154] S. Harris, N. Lamb, and N. Shadbolt, "4store: The design and implementation of a clustered RDF store," in *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pp. 94–109, 2009.

[155] S. Harris, A. Seaborne, and E. Prud'hommeaux, "SPARQL 1.1 Query Language," w3c Reccommendation, World Wide Web Consortium, Mar 2013. Available at `https://www.w3.org/TR/sparql11-query/` [Accessed: 30 Mar 2018].

[156] J. Zhao, C. Goble, R. Stevens, and D. Turi, "Mining Taverna's semantic web of provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 463–472, 2008.

[157] P. Romano, E. Bartocci, G. Bertolini, F. De Paoli, D. Marra, G. Mauri, E. Merelli, and L. Milanesi, "Biowep: A workflow enactment portal for bioinformatics applications," *BMC Bioinformatics*, vol. 8, no. 1, p. S19, 2007.

[158] E. Bartocci, F. Corradini, E. Merelli, and L. Scortichini, "BioWMS: A web-based workflow management system for bioinformatics," *BMC Bioinformatics*, vol. 8, no. 1, p. S2, 2007.

[159] "Kepler Analytical Repository." Available at: `http://library.kepler-project.org/kepler/` [Accessed 30 Mar 2018].

[160] A. Hildebrandt, A. K. Dehof, A. Rurainski, A. Bertsch, M. Schumann, N. C. Toussaint, A. Moll, D. Stöckel, S. Nickels, S. C. Mueller, *et al.*, "BALL – biochemical algorithms library 1.3," *BMC Bioinformatics*, vol. 11, no. 1, p. 531, 2010.

[161] B. Chapman and J. Chang, "Biopython: Python tools for computational biology," *ACM Sigbio Newsletter*, vol. 20, no. 2, pp. 15–19, 2000.

[162] C. Steinbeck, Y. Han, S. Kuhn, O. Horlacher, E. Luttmann, and E. Willighagen, "The Chemistry Development Kit (CDK): An open-source Java library for chemo-and bioinformatics," *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 2, pp. 493–500, 2003.

[163] A. K. Hildebrandt, D. Stöckel, N. M. Fischer, L. de la Garza, J. Krüger, S. Nickels, M. Röttig, C. Schärfe, M. Schumann, P. Thiel, *et al.*, "Ballaxy: Web services for structural bioinformatics," *Bioinformatics*, vol. 31, no. 1, pp. 121–122, 2014.

[164] O. Kohlbacher and H.-P. Lenhof, "BALL – rapid software prototyping in computational molecular biology," *Bioinformatics*, vol. 16, no. 9, pp. 815–824, 2000.

[165] P. J. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, *et al.*, "Biopython: Freely available

python tools for computational molecular biology and bioinformatics," *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, 2009.

[166] C. Steinbeck, C. Hoppe, S. Kuhn, M. Floris, R. Guha, and E. L. Willighagen, "Recent developments of the chemistry development kit (CDK)-an open-source java library for chemo-and bioinformatics," *Current Pharmaceutical Design*, vol. 12, no. 17, pp. 2111–2120, 2006.

[167] T. Kuhn, E. L. Willighagen, A. Zielesny, and C. Steinbeck, "CDK-Taverna: An open workflow environment for cheminformatics," *BMC Bioinformatics*, vol. 11, no. 1, p. 159, 2010.

[168] J. Woodcock and J. Davies, *Using Z: Specification, Refinement, and Proof*. Prentice Hall International, 1996.

[169] J. M. Spivey, "The Z notation: A reference manual," tech. rep., Oxford: Oriel College, 1998. Available at: `https://www.cse.buffalo.edu/LRG/CSE705/Papers/Z-Ref-Manual.pdf` [Accessed 3 Apr 2018].

[170] "Information technology – Z formal specification notation – syntax, type system and semantics," Standard, International Organization for Standardization / International Electrotechnical Commission, Geneva, CH, Aug. 2002.

[171] R. Owen, S. McKeever, J. Davies, and A. Garfinkel, "Toward provably correct models of ventricular cell function," in *Computers in Cardiology, 2006*, pp. 657–660, IEEE, 2006.

[172] J. Arshad, G. Terstyanszky, T. Kiss, and N. Weingarten, "A definition and analysis of the role of meta-workflows in workflow interoperability," in *Science Gateways (IWSG), 2015 7th International Workshop on*, pp. 8–15, IEEE, 2015.

[173] J. Arshad, G. Terstyanszky, T. Kiss, N. Weingarten, and G. Taffoni, "A formal approach to support interoperability in scientific meta-workflows," *Journal of Grid Computing*, vol. 14, no. 4, pp. 655–671, 2016.

[174] L. Freitas and P. Watson, "Formalizing workflows partitioning over federated clouds: Multi-level security and costs," *International Journal of Computer Mathematics*, vol. 91, no. 5, pp. 881–906, 2014.

[175] M. d'Inverno and J. Prophet, "Multidisciplinary investigation into adult stem cell behavior," in *Transactions on Computational Systems Biology III*, pp. 49–64, Springer, 2005.

[176] J. Woodcock, S. Stepney, D. Cooper, J. Clark, and J. Jacob, "The certification of the Mondex electronic purse to ITSEC level E6," *Formal Aspects of Computing*, vol. 20, no. 1, pp. 5–19, 2008.

[177] J. McDermott and L. Freitas, "A formal security policy for xenon," in *Proceedings of the 6th ACM workshop on Formal methods in security engineering*, pp. 43–52, ACM, 2008.

[178] V. Navale and P. E. Bourne, "Cloud computing applications for biomedical science: A perspective," *PLoS Computational Biology*, vol. 14, no. 6, p. e1006144, 2018.

[179] T. Kiss, P. Greenwell, H. Heindl, G. Terstyanszky, and N. Weingarten, "Parameter sweep workflows for modelling carbohydrate recognition," *Journal of Grid Computing*, vol. 8, no. 4, pp. 587–601, 2010.

[180] A. Conesa, S. Götz, J. M. García-Gómez, J. Terol, M. Talón, and M. Robles, "Blast2GO: A universal tool for annotation, visualization and analysis in functional genomics research," *Bioinformatics*, vol. 21, no. 18, pp. 3674–3676, 2005.

[181] M. Kutmon, M. P. van Iersel, A. Bohler, T. Kelder, N. Nunes, A. R. Pico, and C. T. Evelo, "PathVisio 3: An extendable pathway analysis toolbox," *PLoS Computational Biology*, vol. 11, no. 2, p. e1004085, 2015.

[182] D. Schapiro, H. W. Jackson, S. Raghuraman, J. R. Fischer, V. R. Zanotelli, D. Schulz, C. Giesen, R. Catena, Z. Varga, and B. Bodenmiller, "histoCAT: Analysis of cell phenotypes and interactions in multiplex image cytometry data," *Nature Methods*, vol. 14, no. 9, p. 873, 2017.

[183] H. Visti, "Application deployment in CloudBroker: Best practices." Internal report, Nov 2014.

[184] D. Temelkovski, "Extension of Raccoon2 source-code on GitHub," May 2017. Available at `https://github.com/damjanmk/Raccoon2` [Accessed: 1 Apr 2018].

[185] ZINC12, "Subset of 130,216 molecules with molecular weight smaller than 190." Available at `http://zinc.docking.org/db/bysubset/7/7_t90.smi` [Accessed: 2 Apr 2018].

[186] H. Heindl, "Selection of small molecule ligands for in silico docking experiments." Internal report, Nov 2015.

[187] Scripps Research Institute, "AutoDock Vina manual – FAQ." Available at `http://vina.scripps.edu/manual.html#faq` [Accessed: 2 Apr 2018].

[188] CloudSigma, "CloudSigma – Pricing." Available at `https://www.cloudsigma.com/pricing/` [Accessed: 2 Apr 2018].

[189] P. Kacsuk, J. Kovacs, Z. Farkas, A. C. Marosi, G. Gombas, and Z. Balaton, "SZTAKI desktop grid (SZDG): A flexible and scalable desktop grid system," *Journal of Grid Computing*, vol. 7, no. 4, p. 439, 2009.

[190] QSR International Pty Ltd, "NVivo qualitative data analysis software (Version 10)," 2012. Available at: `http://www.qsrinternational.com/nvivo/support-overview/downloads` [Accessed 17 Jul 2018].

[191] A. Bortolato, M. Fanton, J. S. Mason, and S. Moro, "Molecular docking methodologies," in *Biomolecular Simulations*, pp. 339–360, Springer, 2013.

[192] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, "CHARMM: A program for macromolecular energy, minimization, and dynamics calculations," *Journal of Computational Chemistry*, vol. 4, no. 2, pp. 187–217, 1983.

[193] N. Guex and M. C. Peitsch, "SWISS-MODEL and the Swiss-Pdb Viewer: An environment for comparative protein modeling," *Electrophoresis*, vol. 18, no. 15, pp. 2714–2723, 1997.

[194] T. Schwede, J. Kopp, N. Guex, and M. C. Peitsch, "SWISS-MODEL: An automated protein homology-modeling server," *Nucleic Acids Research*, vol. 31, no. 13, pp. 3381–3385, 2003.

[195] H. E. Pence and A. Williams, "ChemSpider: An online chemical information resource," 2010.

[196] K. R. Cousins, "Computer review of ChemDraw Ultra 12.0," 2011. Available: `https://pubs.acs.org/doi/abs/10.1021/ja204075s` [Accessed 30 Mar 2018].

[197] P. Csizmadia, "MarvinSketch and MarvinView: Molecule applets for the World Wide Web," in *Proceedings of ECSOC-3, the third international electronic conference on synthetic organic chemistry*, pp. 367–369, 1999.

[198] M. D. Hanwell, D. E. Curtis, D. C. Lonie, T. Vandermeersch, E. Zurek, and G. R. Hutchison, "Avogadro: An advanced semantic chemical editor, visualization, and analysis platform," *Journal of Cheminformatics*, vol. 4, no. 1, p. 17, 2012.

[199] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin, "UCSF Chimera – a visualization system for exploratory research and analysis," *Journal of Computational Chemistry*, vol. 25, no. 13, pp. 1605–1612, 2004.

[200] C. D. Lau, M. J. Levesque, S. Chien, S. Date, and J. H. Haga, "ViewDock TDW: High-throughput visualization of virtual screening results," *Bioinformatics*, vol. 26, no. 15, pp. 1915–1917, 2010.

[201] K. Chodorow, *MongoDB: The Definitive Guide*. "O'Reilly Media, Inc.", 2013.

[202] J. J. Irwin, B. K. Shoichet, M. M. Mysinger, N. Huang, F. Colizzi, P. Wassam, and Y. Cao, "Automated docking screens: A feasibility study," *Journal of Medicinal Chemistry*, vol. 52, no. 18, pp. 5712–5720, 2009.

[203] X. Zhang, S. E. Wong, and F. C. Lightstone, "Toward fully automated high performance computing drug discovery: A massively parallel virtual screening pipeline for docking and molecular mechanics/generalized Born surface area rescoring to improve enrichment," 2014.

[204] S. Genheden and U. Ryde, "The MM/PBSA and MM/GBSA methods to estimate ligand-binding affinities," *Expert Opinion on Drug Discovery*, vol. 10, no. 5, pp. 449–461, 2015.

[205] X. Zhang, S. E. Wong, and F. C. Lightstone, "Message passing interface and multithreading hybrid for parallel molecular docking of large databases on petascale high performance computing machines," *Journal of Computational Chemistry*, vol. 34, no. 11, pp. 915–927, 2013.

[206] L. Xie, T. Evangelidis, L. Xie, and P. E. Bourne, "Drug discovery using chemical systems biology: Weak inhibition of multiple kinases may contribute to the anticancer effect of nelfinavir," *PLoS Computational Biology*, vol. 7, no. 4, p. e1002037, 2011.

[207] J. Ren, L. Xie, W. W. Li, and P. E. Bourne, "SMAP-WS: A parallel web service for structural proteome-wide ligand-binding site comparison," *Nucleic Acids Research*, vol. 38, no. suppl_2, pp. W441–W444, 2010.

[208] A. N. Jain, "Surflex: Fully automatic flexible molecular docking using a molecular similarity-based search engine," *Journal of Medicinal Chemistry*, vol. 46, no. 4, pp. 499–511, 2003.

[209] Z. Zsoldos, D. Reid, A. Simon, S. B. Sadjad, and A. P. Johnson, "eHiTS: A new fast, exhaustive flexible ligand docking system," *Journal of Molecular Graphics and Modelling*, vol. 26, no. 1, pp. 198–212, 2007.

[210] E. Glaab, "Building a virtual ligand screening pipeline using free software: A survey," *Briefings in Bioinformatics*, vol. 17, no. 2, pp. 352–366, 2015.

[211] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, "Open Babel: An open chemical toolbox," *Journal of Cheminformatics*, vol. 3, no. 1, p. 33, 2011.

[212] P. D'Ursi, F. Chiappori, I. Merelli, P. Cozzi, E. Rovida, and L. Milanesi, "Virtual screening pipeline and ligand modelling for H5N1 neuraminidase," *Biochemical and Biophysical Research Communications*, vol. 383, no. 4, pp. 445–449, 2009.

[213] A. C. Wallace, R. A. Laskowski, and J. M. Thornton, "LIGPLOT: A program to generate schematic diagrams of protein-ligand interactions," *Protein Engineering, Design and Selection*, vol. 8, no. 2, pp. 127–134, 1995.

[214] Z. Farkas, P. Kacsuk, T. Kiss, P. Borsody, Á. Hajnal, Á. Balaskó, and K. Karóczkai, "Autodock gateway for molecular docking simulations in cloud systems," *Cloud Computing with E-science Applications*, p. 300, 2015.

[215] V. B. Chen, W. B. Arendall, J. J. Headd, D. A. Keedy, R. M. Immormino, G. J. Kapral, L. W. Murray, J. S. Richardson, and D. C. Richardson, "MolProbity: All-atom structure validation for macromolecular crystallography," *Acta Crystallographica Section D: Biological Crystallography*, vol. 66, no. 1, pp. 12–21, 2010.

[216] V. B. Chen, I. W. Davis, and D. C. Richardson, "KiNG (Kinemage, Next Generation): A versatile interactive molecular and scientific visualization program," *Protein Science*, vol. 18, no. 11, pp. 2403–2409, 2009.

[217] A. Tsaregorodtsev, V. Garonne, and I. Stokes-Rees, "DIRAC: A scalable lightweight architecture for high throughput computing," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp. 19–25, IEEE Computer Society, 2004.

[218] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario, "The XtreemFS architecture–a case for object-based file systems in grids," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 17, pp. 2049–2060, 2008.

[219] D. W. Erwin and D. F. Snelling, "UNICORE: A grid computing environment," in *European Conference on Parallel Processing*, pp. 825–834, Springer, 2001.

[220] M. C. Burger, "ChemDoodle web components: HTML5 toolkit for chemical graphics, interfaces, and informatics," *Journal of Cheminformatics*, vol. 7, no. 1, p. 35, 2015.

[221] A. Roy, B. Srinivasan, and J. Skolnick, "PoLi: A virtual screening pipeline based on template pocket and ligand similarity," *Journal of Chemical Information and Modeling*, vol. 55, no. 8, pp. 1757–1770, 2015.

[222] H. Zhou and J. Skolnick, "Template-based protein structure modeling using TASSER^{VMT}," *Proteins: Structure, Function, and Bioinformatics*, vol. 80, no. 2, pp. 352–361, 2012.

[223] J. A. Capra, R. A. Laskowski, J. M. Thornton, M. Singh, and T. A. Funkhouser, "Predicting protein ligand binding sites by combining evolutionary sequence conservation and 3D structure," *PLoS Computational Biology*, vol. 5, no. 12, p. e1000585, 2009.

[224] M. Gao and J. Skolnick, "APoc: Large-scale identification of similar protein pockets," *Bioinformatics*, vol. 29, no. 5, pp. 597–604, 2013.

[225] T. A. Wassenaar, M. Van Dijk, N. Loureiro-Ferreira, G. Van Der Schot, S. J. De Vries, C. Schmitz, J. Van Der Zwan, R. Boelens, A. Giachetti, L. Ferella, *et al.*, "WeNMR: Structural biology on the grid," *Journal of Grid Computing*, vol. 10, no. 4, pp. 743–767, 2012.

[226] C. Dominguez, R. Boelens, and A. M. Bonvin, "HADDOCK: A protein- protein docking approach based on biochemical or biophysical information," *Journal of the American Chemical Society*, vol. 125, no. 7, pp. 1731–1737, 2003.

[227] P. Kunszt, L. Blum, B. Hullár, E. Schmid, A. Srebniak, W. Wolski, B. Rinn, F.-J. Elmer, C. Ramakrishnan, A. Quandt, *et al.*, "iPortal: The swiss grid proteomics portal: Requirements and new features based on experience and usability considerations," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 433–445, 2015.

[228] A. Bauch, I. Adamczyk, P. Buczek, F.-J. Elmer, K. Enimanev, P. Glyzewski, M. Kohler, T. Pylak, A. Quandt, C. Ramakrishnan, *et al.*, "openBIS: A flexible framework for managing and analyzing complex data in biology research," *BMC Bioinformatics*, vol. 12, no. 1, p. 468, 2011.

[229] The UniProt Consortium, "UniProt: The universal protein knowledge base," *Nucleic Acids Research*, vol. 45, no. D1, pp. D158–D169, 2017.

[230] M. Fowler, *UML distilled: A brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[231] D. Prandi, "A formal approach to molecular docking," in *International Conference on Computational Methods in Systems Biology*, pp. 78–92, Springer, 2006.

[232] Community Z Tools Project, "CZT for Eclpise." Available at: `http://czt.sourceforge.net/eclipse/` [Accessed 14 Aug 2018].

[233] A. Cockburn, *Agile software development: The cooperative game*. Pearson Education, 2 ed., 2006.

[234] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, *et al.*, "Manifesto for agile software development," 2001.

[235] A. Cockburn, *Crystal clear: A human-powered methodology for small teams*. Pearson Education, 2004.

[236] M. Hellkamp, "Bottle: Python Web Framework – Bottle 0.13-dev documentation," Apr 2018. Available at `https://bottlepy.org/docs/dev/` [Accessed: 1 Apr 2018].

[237] D. Temelkovski, "Implementation of 3 scenarios, source-code on GitHub." Available at `https://github.com/damjanmk/mdrr-scenarios` [Accessed: 2 Sep 2018].

[238] Mike Dirolf (mdirolf), Jeff Jenkins (jeffjenkins), Jim Jones, *et al.*, "PyMongo 3.6.1 documentation – PyMongo 3.6.1 documentation." Available at `https://api.mongodb.com/python/current/` [Accessed: 2 Apr 2018].

[239] N. M. O'Boyle, C. Morley, and G. R. Hutchison, "Pybel: A Python wrapper for the OpenBabel cheminformatics toolkit," *Chemistry Central Journal*, vol. 2, no. 1, p. 5, 2008.

[240] Y. Wang, S. H. Bryant, T. Cheng, J. Wang, A. Gindulyte, B. A. Shoemaker, P. A. Thiessen, S. He, and J. Zhang, "PubChem BioAssay: 2017 update," *Nucleic Acids Research*, vol. 45, no. D1, pp. D955–D963, 2016.

[241] S. Kim, P. A. Thiessen, E. E. Bolton, and S. H. Bryant, "PUG-SOAP and PUG-REST: Web services for programmatic access to chemical information in PubChem," *Nucleic Acids Research*, vol. 43, no. W1, pp. W605–W611, 2015.

[242] PubChem, "PubChem docs – programmatic access – request volume limitations." Available at `http://pubchemdocs.ncbi.nlm.nih.gov/programmatic-access$_RequestVolumeLimitations` [Accessed: 2 Apr 2018].

[243] D. S. Goodsell, S. Dutta, C. Zardecki, M. Voigt, H. M. Berman, and S. K. Burley, "The RCSB PDB "molecule of the month": Inspiring a molecular view of biology," *PLoS Biology*, vol. 13, no. 5, p. e1002140, 2015.

[244] ZINC15, "Subset of molecules approved in major jurisdictions." Available at `http://zinc15.docking.org/substances/subsets/world/` [Accessed: 2 Apr 2018].

[245] O. Trott, "Vina video tutorial," Feb 2014. Available at `http://vina.scripps.edu/tutorial.html` [Accessed: 2 Apr 2018].